Efficient Lagrangian Relaxation Algorithms for Exact Inference in Natural Language Tasks

by

Alexander M. Rush

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Science in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2011

Author .....
Department of Electrical Engineering and Computer Science
May 20, 2011

Certified by .....
Michael Collins
Professor
Thesis Supervisor

Accepted by .....
Leslie A. Kolodziejski
Chair, Department Committee on Graduate Theses

Efficient Lagrangian Relaxation Algorithms for Exact Inference in Natural Language Tasks

by

Alexander M. Rush

Submitted to the Department of Electrical Engineering and Computer Science
on May 20, 2011, in partial fulfillment of the
requirements for the degree of
Master of Science in Computer Science and Engineering

Abstract

For many tasks in natural language processing, finding the best solution requires a search over a large set of possible structures. Solving these combinatorial search problems exactly can be inefficient, and so researchers often use approximate techniques at the cost of model accuracy. In this thesis, we turn to Lagrangian relaxation as an alternative to approximate inference in natural language tasks. We demonstrate that Lagrangian relaxation algorithms provide efficient solutions while still maintaining formal guarantees. The approach leads to inference algorithms with the following properties:

- The resulting algorithms are simple and efficient, building on standard combinatorial algorithms for relaxed problems.
- The algorithms provably solve a linear programming (LP) relaxation of the original inference problem.
- Empirically, the relaxation often leads to an exact solution to the original problem.

We develop Lagrangian relaxation algorithms for several important tasks in natural language processing including higher-order non-projective dependency parsing, syntactic machine translation, integrated constituency and dependency parsing, and part-of-speech tagging with inter-sentence constraints. For each of these tasks, we show that the Lagrangian relaxation algorithms are often significantly faster than exact methods while finding the exact solution with a certificate of optimality in the vast majority of examples.

Thesis Supervisor: Michael Collins
Title: Professor

Contents

Chapter 1

Combinatorial Optimization and Lagrangian Relaxation

1.1 Introduction

For many tasks in natural language processing, finding the best solution requires a search over a large set of possible structures. Solving these combinatorial search problems exactly can be inefficient, and so researchers often use approximate techniques at the cost of model accuracy. In this thesis, we turn to Lagrangian relaxation as an alternative to approximate inference in natural language tasks. We demonstrate that Lagrangian relaxation algorithms provide efficient solutions while still maintaining formal guarantees. The approach leads to inference algorithms with the following properties:

- The resulting algorithms are simple and efficient, building on standard combinatorial algorithms for relaxed problems.
- The algorithms provably solve a linear programming (LP) relaxation of the original inference problem.
- Empirically, the relaxation often leads to an exact solution to the original problem.

The central abstraction used throughout this work is the decoding problem for structured prediction. We assume that we have a set $\mathcal{Y}$ of possible linguistic structures, for instance taggings or parses of a sentence, and a function f that scores each structure in the set.

Decoding is the optimization problem:

$$y^* = \arg\max_{y \in \mathcal{Y}} f(y)$$

Thoughout this thesis we will look at methods to solve variants of this optimization problem relevant to different areas of NLP.

Solving the decoding problem exactly can be quite difficult for important natural language tasks. In certain cases, this optimization problem is NP-hard. For instance in Chapter 4, we explore the problem of higher-order non-projective dependency parsing, an important model for syntax that is NP-hard to solve. For other problems, decoding algorithms are polynomial time but have a large exponent that make them intractable for practical use. For these problems, Lagrangian relaxation can offer significant speed improvements while still maintaining formal guarantees.

Each of the algorithms presented in this thesis follows a similar form. We start with an exact, but inefficient algorithm for the task. As an alternative, we propose a related problem that can be solved efficiently but has the potential to produce invalid solutions. We then repeatedly solve modified versions of the problem. We show that if we ever a valid solution, this solution is exact for the original problem.

The benefit of this approach is that it leaves flexibility in choosing the relaxed problem. We can tailor the problem to take advantage of a wide array of simple and fast combinatorial algorithm. In different sections of this thesis, we make use of Viterbi decoding, (directed) minimum spanning tree, all-pairs shortest path, and A* search to solve relaxations. This freedom also allows us to use knowledge of the underlying NLP problems to better optimize these combinatorial algorithms.

Before exploring the use of Lagrangian relaxation for solving decoding problems in NLP, we present the seminal work of Held and Karp on the use of relaxation as a method for solving NP-hard combinatorial optimization problems.

1.2 Traveling Salesman Relaxation

In this section, we describe the Lagrangian relaxation algorithm of Held and Karp [1971] for the symmetric traveling salesman problem (TSP). While we do not directly utilize this relaxation, its simplicity and efficiency demonstrate the usefulness of Lagrangian relaxation

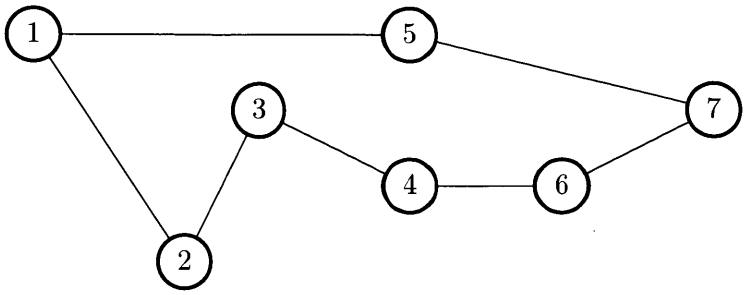


Figure 1-1: A tour over G

and motivates its use in our work.

The algorithm follows the relaxation strategy we outlined in the introduction. Since TSP is NP-hard, we cannot hope to efficiently solve it directly. Instead, we propose a simpler problem, minimum 1-tree, that is close to the original problem but can produce invalid solutions. We then show that by solving variants of the minimum 1-tree problem we can produce lower bounds of the original problem. Furthermore, we show that if we find valid solution to TSP in the process, the solution is exact.

Formally, the symmetric traveling salesman problem is a minimization problem over the set of weighted tours in an undirected graph. We assume an undirected graph $G = (V, E)$ with the nodes $V = \{1, \ldots, n\}$. The graph has a weight function on edges $w : E \to \mathcal{R}$. A tour of the graph is a cyclic path starting at 1 that visits each node exactly once. Figure 1.2 gives an example tour.

The tours of the graph are given by the set of binary vectors:

$$\mathcal{Y} = \{y \in \{0,1\}^{|E|} : \forall i, \sum_{j>i} y(i,j) + \sum_{j<i} y(j,i) = 2, \quad (1.1)$$

$$\forall S \subset V \setminus \{1\}, \sum_{i<j \in S} y(i,j) \leq |S| - 1, \quad (1.2)$$

$$\forall i, \sum_{(i,j) \in E} y(i,j) = n\} \quad (1.3)$$

The element $y(i,j) = 1$ if edge (i,j) is included in the tour. Constraint 1.1 says that the tour must enter and leave each node in the graph. Constraint 1.2 is a no cycle constraint. It says there can be no cycles in any subgraph of $V \setminus \{1\}$. Constraint 1.3 enforces that a tour has n nodes. This constraint is implied by the first constraint, but it will be useful in the relaxation.

To score sets of edges, we introduce a function, $f : \{0,1\}^{|E|} \rightarrow \mathcal{R}$, over binary vectors of edges. The scoring function simply calculates the total distance of all edges:

$$f(y) = \sum_{(i,j) \in E} y(i,j) w(i,j)$$

The TSP problem is to find the minimum tour by this scoring function,

$$y^* = \arg\min_{y \in \mathcal{Y}} f(y)$$

Note that calculating $f(y)$ is very efficient but the set of tours has size $|\mathcal{Y}| = O(2^n)$. Since TSP is NP-hard, there is unlikely to be a polynomial time algorithm for this optimization.

Held and Karp propose a different strategy. Instead of directly using $\mathcal{Y}$ they construct a relaxed problem over $\mathcal{Z}$ where $\mathcal{Y} \subset \mathcal{Z}$ and solve:

$$z^* = \arg\min_{z \in \mathcal{Z}} f(z)$$

With the right choice of $\mathcal{Z}$ this optimization is very efficient, and can be used to solve the original problem.

The set $\mathcal{Z}$ they propose is the set of 1-trees of the graph, a set very similar to the set of spanning trees. A spanning tree is a set of edges that includes each node at least once and has no cycles. The set of spanning trees is:

$$SPAN = \{z \in \{0,1\}^{|E|} : \forall S \subset V, \sum_{i<j \in S} z(i,j) \leq |S| - 1, \tag{1.4}$$

$$\forall i, \sum_{(i,j) \in E} z(i,j) = n - 1 \} \tag{1.5}$$

Constraint 1.4 is the no cycle constraint applied to all subgraphs of G. Constraint 1.5 is a counting constraint that says $n-1$ nodes are on. We can find the minimum spanning tree of a graph by running a greedy algorithm over edges such of Prim's algorithm.

A 1-tree is a slight modification of a spanning tree. It consists of a spanning tree over the subgraph $V \setminus \{1\}$ as well as two additional edges incident to node 1. A 1-tree can be a spanning tree, but it can also include at most one cycle. Figure 1.2 gives an example of a

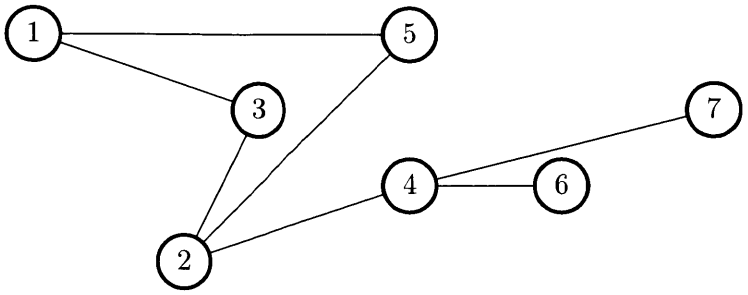


Figure 1-2: A 1-tree over G

1-tree in G. The formal set is:

$$\begin{aligned} \mathcal{Z} = \{z \in \{0,1\}^{|E|} : \forall S \subset V \setminus \{1\}, \sum_{i<j \in S} z(i,j) &\leq |S| - 1 \\ \forall i, \sum_{(i,j) \in E} z(i,j) &= n\} \end{aligned}$$

Unlike for TSP, we have an efficient algorithm for finding the minimum 1-tree. We do this by using the greedy algorithm for the minimum spanning tree over $V \setminus \{1\}$, and then choosing two edges incident to 1.

The minimum 1-tree problem is a relaxation of the TSP. We will discuss the formal details of the general relaxation technique in Chapter 2. For now we walk through the derivation.

We can derive the relaxation directly from our original formulation of the problem,

$$y^* = \arg\min_{y \in \mathcal{Y}} f(y)$$

The set of tours $\mathcal{Y}$ in this minimization is defined by three constraints. The only constraint for tours that is not a constraint for 1-trees is that each node must be incident to two edges:

$$\forall i, \sum_{j>i} y(i,j) + \sum_{j<i} y(j,i) = 2$$

Without this constraint we are left with an optimization over $\mathcal{Z}$. We therefore remove this constraint by relaxing it to form the Lagrangian dual. This gives as a new minimization

Inputs: $G = (V, E)$ graph, w set of weights, α update rate
Returns: if successful, z the minimum tour
$u^{(1)} \leftarrow 0$
for $k = 1$ to K **do**
$z^{(k)} \leftarrow \arg\min_{z \in \mathcal{Z}} \sum_{(i,j) \in E} z(i,j)(w(i,j) - u^{(k)}(i) - u^{(k)}(j))$
if $z^{(k)} \in \mathcal{Y}$ **then**
return $z^{(k)}$
else
$u^{(k+1)}(i) \leftarrow u^{(k)} + \alpha_k(2 - (\sum_{j>i} z^{(k)}(i,j) + \sum_{j<i} z^{(k)}(j,i)))$

Figure 1-3: Lagrangian relaxation for TSP

over $\mathcal{Z}$ as well as a Lagrange multiplier u for the constraints on each node i:

$$\begin{aligned} L(u) &= \min_{z \in \mathcal{Z}} f(z) + \sum_i u(i)(2 - (\sum_{j>i} z(i,j) + \sum_{j<i} z(j,i))) \\ &= 2\sum_i u(i) + \min_{z \in \mathcal{Z}} f(z) - \sum_i \sum_{j>i} u(i)z(i,j) - \sum_i \sum_{j<i} u(i)z(j,i) \\ &= 2\sum_i u(i) + \min_{z \in \mathcal{Z}} \sum_{(i,j) \in E} z(i,j)(w(i,j) - u(i) - u(j)) \end{aligned}$$

After rearranging terms, we are left with a constant term and a minimum 1-tree problem with weights modified by u. Since this minimization is efficient to compute, $L(u)$ is efficient to calculate for any value of u.

Furthermore we know by weak duality that for any value of u the dual $L(u)$ is a lower bound on the optimal solution, $L(u) \leq f(y^*)$. To find the tightest lower bound we maximize $L(u)$:

$$\max_u L(u)$$

We solve this dual maximization problem using subgradient ascent. For now, we present the algorithm with derivation. More details about this approach are given in Chapter 2.

The full algorithm is given in figure 1.2. The general approach is to repeatedly solve the minimum 1-tree problem and use the resulting 1-tree to update the multipliers u. We begin with each multiplier set to zero. At each step, we solve the minimum 1-tree problem to find the best 1-tree $z^{(k)}$. If this happens to be a tour, then the lower bound is tight, and we return that tour. If not, we make an update to our multipliers u based on the current solution.

This algorithm is guaranteed to maximize the lower bound. If the bound is tight, we

will find a $z \in \mathcal{Y}$ which is the exact solution. However, the algorithm is not guaranteed to converge, in which case we will not immediately retreive the exact solution. We will see that for many of the problems in this thesis there are straightforward methods to get an approximate solutions in this case. Furthermore in Chapter 5, we explore methods to search for exact solutions when we do not reach convergence.

1.3 Combinatorial Algorithms in NLP

Combinatorial optimization problems, like the TSP presented in the last section, play central role of many NLP tasks, particularly for the class of problems requiring structured prediction. In a structured prediction problem, we first learn the parameters of the scoring function f and then solve the decoding problem to find the best scoring structure

$$\arg\max_{y \in \mathcal{Y}} f(y)$$

where $\mathcal{Y}$ is some combinatorial set of linguistic structures. Since it is necessary to solve this optimization to utilize the learned model, the efficiency of decoding depends on the choice of the combinatorial algorithm.

For many NLP problems, we can decode by directly using a well-studied combinatorial optimization technique. For instance, Knight [1999] show that decoding certain models of machine translation corresponds to solving a TSP. To perform translation, we first learn a set of translation weights, and then for each sentence, we project these weights to an appropriate graph. The best translation corresponds to the minimum tour over the graph. Figure 1.3 shows a simple version of this mapping.

We will see this same relationship for the NLP tasks presented in this thesis. For instance, in Chapter 4 we introduce the problem of first-order non-projective dependency parsing. Finding the best parse structure under this model requires a similar combinatorial optimization where $\mathcal{Y}$ is the set of directed spanning trees. When we move to more complex models of non-projective dependency parsing, we use maximum directed spanning tree as part of a relaxed problem in a similar way to how Held and Karp use the minimum 1-tree.

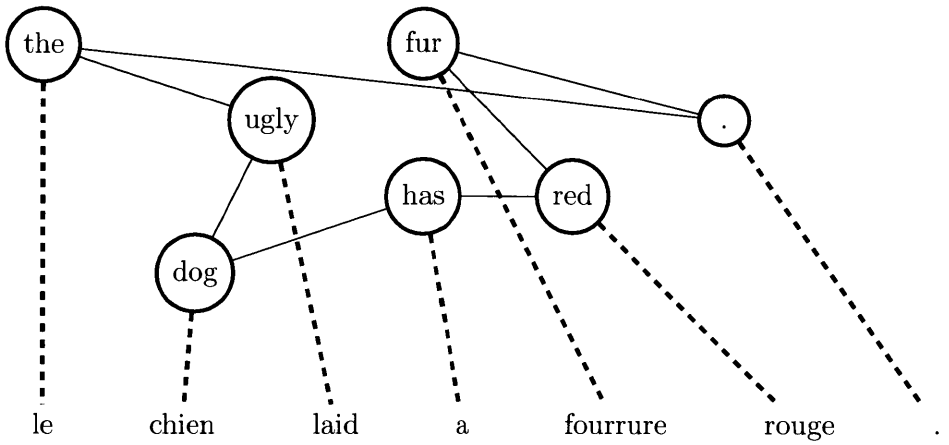


Figure 1-4: A simplified version of translation decoding as a traveling salesman problem. The aim is to translate the sentence "*la chien laid a fourrure rouge.*" Each node in the graph represents a translated word. Distances in the graph are English language model scores. A tour corresponds to translating each word once and assigning an English word order.

1.4 Overview

Given that many NLP decoding tasks can be expressed as combinatorial optimization problems, our goal is to find efficient algorithms to solve these problems. In this thesis, we approach this problem by developing Lagrangian relaxation algorithms targeted to several difficult NLP tasks. We show empirically that these algorithms are efficient and very often find exact solutions in wide-ranging scenarios.

Each chapter of the thesis is devoted to a challenging decoding problem in NLP. For each problem, we motivate the use of Lagrangian relaxation by showing that exact algorithms scale poorly. We then propose a Lagrangian relaxation algorithm that exploits the combinatorial structure of the problem. After developing the formal properties of the algorithm, we show empirically that the new algorithm is simple, efficient, and almost always as optimal as the original exact technique.

In the initial two chapters, we look at relaxation methods for integrating multiple models. In this setup, we assume that we have efficient solvers for certain models, but our goal is to maximize some combination of the individual models. We focus particularly on the applications of part-of-speech tagging and projective forms of parsing as important combinatorial problems important for NLP.

In Chapter 2, we explore the problem of integrating two models solvable by dynamic

programming that share common structure. In these problems, we assume two models with scoring functions f and g that produce different types of output. Our goal is to maximize the sum, $f + g$, while maintaining the shared structure in the output. We derive algorithms for integrated constituency parsing and part-of-speech tagging as well as integrated constituency parsing and dependency parsing.

In Chapter 3, we turn to combinatorial problems that involve an entire corpus of sentences tied together with inter-sentence constraints. These constraints encourage common decisions among sentence-level models. We encode the constraints using a global Markov random field, and experiment with part-of-speech tagging and dependency parsing over the sentence-level models.

The next two chapters focus on algorithms for specific, challenging NLP applications. In these chapters, we look at decoding problems that are difficult to solve in their base form. We describe methods to relax these problems into simple combinatorial subproblems that can be solved efficiently. In addition to dynamic programming, these algorithms make use of minimum-spanning tree algorithms as well as all-pairs shortest path.

Chapter 4 centers on higher-order non-projective dependency parsing. Non-projective parsing is an important variety for dependency parsing for certain languages with non-projective syntax. Models for simple first-order non-projective dependency parsing can be decoded using greedy algorithms to find the maximum directed spanning tree. However, models that include higher-order features are NP-hard to decode. We give a dual decomposition algorithm for higher-order dependency parsing to decodes these models using simple combinatorial algorithms. In practice, this algorithm almost always finds an exact solution to the problem.

In Chapter 5, we consider methods for decoding syntactic translation models. These models are similar to parsing models, but they must also integrate a language model as part of translation. The difficulty is finding an exact solution that maximizes both the language model and the parsing model. We demonstrate a Lagrangian relaxation algorithm that splits the decoding problem into two stages each of which can be solved efficiently. Unlike in previous relaxations, the simple version of this algorithm fails to find an exact solution on many examples. To fix this problem, we use a tightening strategy to add additional constraints to the problem.

We conclude in Chapter 6 by discussing directions for future work.

1.5 Related Work

The ideas presented in this thesis are based on many areas of previous work both in optimization and NLP.

Both dual decomposition and Lagrangian relaxation have a long history in combinatorial optimization. Our work was originally inspired by recent work on dual decomposition for inference in graphical models [Wainwright et al., 2005a, Komodakis et al., 2007]. Other work has made extensive use of decomposition approaches for efficiently solving LP relaxations for graphical models (e.g., Sontag et al. [2008]).

Our work is similar in spirit to methods that incorporate combinatorial solvers within LBP, either for MAP inference [Duchi et al., 2007] or for computing marginals [Smith and Eisner, 2008b]. Our method aims to solve the MAP inference problem, but unlike LBP it has relatively clear convergence guarantees, in terms of solving an LP relaxation.

In other work, variational inference (and linear programming) in graphical models has relied extensively on the idea of marginal polytopes, see for example Wainwright and Jordan [2008]. Several papers have explored LP relaxations as a framework for deriving alternative algorithms to loopy belief propagation (LBP) (e.g., Globerson and Jaakkola [2007] and Sontag et al. [2008]).

Finally recent work has considered approaches for approximate or exact inference for challenging NLP problems, with the aim of improved efficiency: examples include coarse-to-fine inference [Charniak and Johnson, 2005], sampling methods [Finkel et al., 2005], A-Star search [Klein and Manning, 2003, Felzenszwalb and McAllester, 2007], and work on Markov logic networks/SAT solvers [Meza-Ruiz and Riedel, 2009].

Chapter 2

Combining Dynamic Programs

This chapter is joint work with David Sontag, Michael Collins, and Tommi Jaakkola. An earlier version of the content in this chapter was published at EMNLP 2010 [Rush et al., 2010].

2.1 Introduction

The focus of this chapter is on dynamic programming as a method for solving decoding problems. Dynamic programming algorithms have proven remarkably useful in many NLP domains, including sequence labeling, parsing, and machine translation. Unfortunately, as the underlying models for these problems become more complex, for example through the addition of new features or components, dynamic programming algorithms can quickly explode in terms of computational or implementational complexity.

To circumvent the complexity cost of using exact dynamic programming, we develop a Lagrangian relaxation algorithm for extending dynamic programming approaches in NLP. We give an algorithm that repeatedly solves a relaxed version of the full dynamic programming problem and show empirically that this technique very often finds an exact solution to the original problem.

Our algorithm is based on *dual decomposition*, an important special case of Lagrangian relaxation that we utilize for several problems in this thesis. Dual decomposition leverages the observation that complex inference problems can often be decomposed into efficiently solvable subproblems. Our relaxed problem is to just solve each of these subproblems independently.

The structure of this chapter is as follows. We first give two examples as an illustration of the approach: 1) integrated parsing and trigram POS tagging; and 2) combined phrase-structure and dependency parsing. In both settings, it is possible to solve the integrated problem through an "intersected" dynamic program (e.g., for integration of parsing and tagging, the construction from Bar-Hillel et al. [1964] can be used). However, these approaches are less computationally efficient than our algorithms, and are considerably more complex to implement.

We then give guarantees of convergence for the example algorithms. We describe LP relaxations for the two problems, building on connections between dynamic programming algorithms and marginal polytopes, as described in Martin et al. [1990]. We then show that the example algorithms are instantiations of Lagrangian relaxation a general method for solving linear programs of a particular form.

Finally, we describe experiments. First, we consider the integration of the generative model for phrase-structure parsing of Collins [2003], with the second-order discriminative dependency parser of Koo et al. [2008]. This is an interesting problem in its own right: the goal is to inject the high performance of discriminative dependency models into phrase-structure parsing. The method uses off-the-shelf decoders for the two models. We find three main results: 1) in spite of solving an LP relaxation, empirically the method finds an exact solution on over 99% of the examples; 2) the method converges quickly, typically requiring fewer than 10 iterations of decoding; 3) the method gives gains over a baseline method that enforces exact agreement on dependencies between the phrase-structure parser and the dependency parser (our re-implementation of Collins [2003] has an F1 score of 88.1%; the baseline method has an F1 score of 89.7%; and the dual decomposition method has an F1 score of 90.7%).

In a second set of experiments, we use dual decomposition to integrate the trigram POS tagger of Toutanova and Manning [2000] with the parser of Collins [2003]. We again find that the method finds an exact solution in almost all cases, with convergence in just a few iterations of decoding.

2.2 Background: Structured Models for NLP

We begin by describing the type of models used throughout the chapter. We take some care to set up notation that will allow us to make a clear connection between inference problems and linear programming.

Our first example is weighted CFG parsing. We assume a context-free grammar, in Chomsky normal form, with a set of non-terminals N. The grammar contains all rules of the form $A \rightarrow B\ C$ and $A \rightarrow w$ where $A, B, C \in N$ and $w \in V$ (it is simple to relax this assumption to give a more constrained grammar). For rules of the form $A \rightarrow w$ we refer to A as the part-of-speech (POS) tag for w. We allow any non-terminal to be at the root of the tree.

Given a sentence with n words, $w_1, w_2, \ldots w_n$, a parse tree is a set of rule productions of the form $\langle A \rightarrow B\ C, i, k, j \rangle$ where $A, B, C \in N$, and $1 \leq i \leq k < j \leq n$. Each rule production represents the use of CFG rule $A \rightarrow B\ C$ where non-terminal A spans words $w_i \ldots w_j$, non-terminal B spans words $w_i \ldots w_k$, and non-terminal C spans words $w_{k+1} \ldots w_j$. There are $O(|N|^3 n^3)$ such rule productions. Each parse tree corresponds to a subset of these rule productions, of size $n - 1$, that forms a well-formed parse tree.[1]

We now define the *index set* for CFG parsing as

$$\mathcal{I} = \{\langle A \rightarrow B\ C, i, k, j \rangle :\ A, B, C \in N,\ 1 \leq i \leq k < j \leq n\}$$

Each parse tree is a vector $y = \{y_r : r \in \mathcal{I}\}$, with $y_r = 1$ if rule r is in the parse tree, and $y_r = 0$ otherwise. Hence each parse tree is represented as a vector in $\{0, 1\}^m$, where $m = |\mathcal{I}|$. We use $\mathcal{Y}$ to denote the set of all valid parse-tree vectors; the set $\mathcal{Y}$ is a subset of $\{0, 1\}^m$ (not all binary vectors correspond to valid parse trees).

In addition, we assume a vector $\theta = \{\theta_r : r \in \mathcal{I}\}$ that specifies a weight for each rule production.[2] Each θ_r can take any value in the reals. The optimal parse tree is $y^* = \arg\max_{y \in \mathcal{Y}}\ y \cdot \theta$ where $y \cdot \theta = \sum_r y_r \theta_r$ is the inner product between y and θ.

We use y_r and $y(r)$ interchangeably (similarly for θ_r and $\theta(r)$) to refer to the r'th

[1] We do not require rules of the form $A \rightarrow w_i$ in this representation, as they are redundant: specifically, a rule production $\langle A \rightarrow B\ C, i, k, j \rangle$ implies a rule $B \rightarrow w_i$ iff $i = k$, and $C \rightarrow w_j$ iff $j = k + 1$.

[2] We do not require parameters for rules of the form $A \rightarrow w$, as they can be folded into rule production parameters. E.g., under a PCFG we define $\theta(A \rightarrow B\ C, i, k, j) = \log P(A \rightarrow B\ C \mid A) + \delta_{i,k} \log P(B \rightarrow w_i | B) + \delta_{k+1,j} \log P(C \rightarrow w_j | C)$ where $\delta_{x,y} = 1$ if $x = y$, 0 otherwise.

component of the vector y. For example $\theta(A \rightarrow B\ C, i, k, j)$ is a weight for the rule $\langle A \rightarrow B\ C, i, k, j \rangle$.

We will use similar notation for other problems. As a second example, in POS tagging the task is to map a sentence of n words $w_1 \ldots w_n$ to a tag sequence $t_1 \ldots t_n$, where each t_i is chosen from a set T of possible tags. We assume a trigram tagger, where a tag sequence is represented through decisions $\langle (A, B) \rightarrow C, i \rangle$ where $A, B, C \in T$, and $i \in \{3 \ldots n\}$. Each production represents a transition where C is the tag of word w_i, and (A, B) are the previous two tags. The index set for tagging is

$$\mathcal{I}_{\text{tag}} = \{\langle (A, B) \rightarrow C, i \rangle \ : \ A, B, C \in T, 3 \leq i \leq n\}$$

Note that we do not need transitions for $i = 1$ or $i = 2$, because the transition $\langle (A, B) \rightarrow C, 3 \rangle$ specifies the first three tags in the sentence.[3]

Each tag sequence is represented as a vector $z = \{z_r : r \in \mathcal{I}_{\text{tag}}\}$, and we denote the set of valid tag sequences, a subset of $\{0, 1\}^{|\mathcal{I}_{\text{tag}}|}$, as $\mathcal{Z}$. Given a parameter vector $\theta = \{\theta_r : r \in \mathcal{I}_{\text{tag}}\}$, the optimal tag sequence is $\arg\max_{z \in \mathcal{Z}} z \cdot \theta$.

As a modification to the above approach, we will find it convenient to introduce extended index sets for both the CFG and POS tagging examples. For the CFG case we define the extended index set to be $\mathcal{I}' = \mathcal{I} \cup \mathcal{I}_{\text{uni}}$ where

$$\mathcal{I}_{\text{uni}} = \{(i, t) : i \in \{1 \ldots n\}, t \in T\}$$

Here each pair (i, t) represents word w_i being assigned the tag t. Thus each parse-tree vector y will have additional (binary) components $y(i, t)$ specifying whether or not word i is assigned tag t. (Throughout this chapter we will assume that the tag-set used by the tagger, T, is a subset of the set of non-terminals considered by the parser, N.) Note that this representation is over-complete, since a parse tree determines a unique tagging for a sentence: more explicitly, for any $i \in \{1 \ldots n\}$, $Y \in T$, the following linear constraint holds:

$$y(i, Y) = \sum_{k=i+1}^{n} \sum_{X, Z \in N} y(X \rightarrow Y\ Z, i, i, k) + \sum_{k=1}^{i-1} \sum_{X, Z \in N} y(X \rightarrow Z\ Y, k, i-1, i)$$

[3] As one example, in an HMM, the parameter $\theta((A, B) \rightarrow C, 3)$ would be $\log P(A| * *) + \log P(B| * A) + \log P(C|AB) + \log P(w_1|A) + \log P(w_2|B) + \log P(w_3|C)$, where $*$ is the start symbol.

We apply the same extension to the tagging index set, effectively mapping trigrams down to unigram assignments, again giving an over-complete representation. The extended index set for tagging is referred to as $\mathcal{I}'_{\text{tag}}$.

From here on we will make exclusive use of extended index sets for CFG parsing and trigram tagging. We use the set $\mathcal{Y}$ to refer to the set of valid parse structures under the extended representation; each $y \in \mathcal{Y}$ is a binary vector of length $|\mathcal{I}'|$. We similarly use $\mathcal{Z}$ to refer to the set of valid tag structures under the extended representation. We assume parameter vectors for the two problems, $\theta^{\text{cfg}} \in \mathcal{R}^{|\mathcal{I}'|}$ and $\theta^{\text{tag}} \in \mathcal{R}^{|\mathcal{I}'_{\text{tag}}|}$.

2.3 Two Examples

This section describes the dual decomposition approach for two inference problems in NLP.

2.3.1 Integrated Parsing and Trigram Tagging

We now describe the dual decomposition approach for integrated parsing and trigram tagging. First, define the set $\mathcal{Q}$ as follows:

$$\mathcal{Q} = \{(y, z) : y \in \mathcal{Y}, z \in \mathcal{Z}, \\ y(i,t) = z(i,t) \text{ for all } (i,t) \in \mathcal{I}_{uni}\} \tag{2.1}$$

Hence $\mathcal{Q}$ is the set of all (y, z) pairs that agree on their part-of-speech assignments. The integrated parsing and trigram tagging problem is then to solve

$$\max_{(y,z)\in\mathcal{Q}} \left(y \cdot \theta^{\text{cfg}} + z \cdot \theta^{\text{tag}} \right) \tag{2.2}$$

This problem is equivalent to

$$\max_{y\in\mathcal{Y}} \left(y \cdot \theta^{\text{cfg}} + g(y) \cdot \theta^{\text{tag}} \right)$$

where $g : \mathcal{Y} \rightarrow \mathcal{Z}$ is a function that maps a parse tree y to its set of trigrams $z = g(y)$. The benefit of the formulation in Eq. 2.2 is that it makes explicit the idea of maximizing over all pairs (y, z) under a set of agreement constraints $y(i,t) = z(i,t)$—this intuition will be

Set $u^{(1)}(i,t) \leftarrow 0$ for all $(i,t) \in \mathcal{I}_{\text{uni}}$
for $k = 1$ to K **do**

$$y^{(k)} \leftarrow \arg\max_{y \in \mathcal{Y}} \; (y \cdot \theta^{\text{cfg}} - \sum_{(i,t) \in \mathcal{I}_{\text{uni}}} u^{(k)}(i,t) y(i,t))$$

$$z^{(k)} \leftarrow \arg\max_{z \in \mathcal{Z}} \; (z \cdot \theta^{\text{tag}} + \sum_{(i,t) \in \mathcal{I}_{\text{uni}}} u^{(k)}(i,t) z(i,t))$$

  if $y^{(k)}(i,t) = z^{(k)}(i,t)$ for all $(i,t) \in \mathcal{I}_{\text{uni}}$ **then**
    return $(y^{(k)}, z^{(k)})$
  for all $(i,t) \in \mathcal{I}_{\text{uni}}$,
    $u^{(k+1)}(i,t) \leftarrow u^{(k)}(i,t) + \alpha_k (y^{(k)}(i,t) - z^{(k)}(i,t))$
return $(y^{(K)}, z^{(K)})$

Figure 2-1: The algorithm for integrated parsing and tagging. The parameters $\alpha_k > 0$ for $k = 1 \ldots K$ specify step sizes for each iteration. The two arg max problems can be solved using dynamic programming.

central to the algorithms in this chapter.

With this in mind, we note that we have efficient methods for the inference problems of tagging and parsing alone, and that our combined objective almost separates into these two independent problems. In fact, if we drop the $y(i,t) = z(i,t)$ constraints from the optimization problem, the problem splits into two parts, which can each be solved relatively easily using dynamic programming:

$$(y^*, z^*) = (\arg\max_{y \in \mathcal{Y}} y \cdot \theta^{\text{cfg}}, \arg\max_{z \in \mathcal{Z}} z \cdot \theta^{\text{tag}})$$

Dual decomposition exploits this idea; it results in the algorithm given in figure 2-1. The algorithm optimizes the combined objective by repeatedly solving the two sub-problems separately—that is, it directly solves the harder optimization problem using an existing CFG parser and trigram tagger. After each iteration the algorithm adjusts the weights $u(i,t)$; these updates modify the objective functions for the two models, encouraging them to agree on the same POS sequence. In section 2.5.1 we will show that the variables $u(i,t)$ are Lagrange multipliers enforcing agreement constraints, and that the algorithm corresponds to a (sub)gradient method for optimization of a dual function. The algorithm is easy to implement: all that is required is a decoding algorithm for each of the two models, and simple additive updates to the Lagrange multipliers enforcing agreement between the two models.

2.3.2 Integrating Two Lexicalized Parsers

Our second example problem is the integration of a phrase-structure parser with a higher-order dependency parser. The goal is to add higher-order features to phrase-structure parsing without greatly increasing the complexity of inference.

First, we define an index set for second-order unlabeled projective dependency parsing. The second-order parser considers first-order dependencies, as well as grandparent and sibling second-order dependencies (e.g., see Carreras [2007b]). We assume that $\mathcal{I}_{\text{dep}}$ is an index set containing all such dependencies (for brevity we omit the details of this index set). For convenience we define an extended index set that makes explicit use of first-order dependencies, $\mathcal{I}'_{\text{dep}} = \mathcal{I}_{\text{dep}} \cup \mathcal{I}_{\text{first}}$, where

$$\mathcal{I}_{\text{first}} = \{(i, j) \ : \ i \in \{0 \ldots n\}, j \in \{1 \ldots n\}, i \neq j\}$$

Here (i, j) represents a dependency with head w_i and modifier w_j ($i = 0$ corresponds to the root symbol in the parse). We use $\mathcal{D} \subseteq \{0, 1\}^{|\mathcal{I}'_{\text{dep}}|}$ to denote the set of valid projective dependency parses.

The second model we use is a lexicalized CFG. Each symbol in the grammar takes the form $A(h)$ where $A \in N$ is a non-terminal, and $h \in \{1 \ldots n\}$ is an index specifying that w_h is the head of the constituent. Rule productions take the form $\langle A(a) \rightarrow B(b) \ C(c), i, k, j \rangle$ where $b \in \{i \ldots k\}$, $c \in \{(k+1) \ldots j\}$, and a is equal to b or c, depending on whether A receives its head-word from its left or right child. Each such rule implies a dependency (a, b) if $a = c$, or (a, c) if $a = b$. We take $\mathcal{I}_{\text{head}}$ to be the index set of all such rules, and $\mathcal{I}'_{\text{head}} = \mathcal{I}_{\text{head}} \cup \mathcal{I}_{\text{first}}$ to be the extended index set. We define $\mathcal{H} \subseteq \{0, 1\}^{|\mathcal{I}'_{\text{head}}|}$ to be the set of valid parse trees.

The integrated parsing problem is then to find

$$(y^*, d^*) = \arg \max_{(y,d) \in \mathcal{R}} \left(y \cdot \theta^{\text{head}} + d \cdot \theta^{\text{dep}} \right) \tag{2.3}$$

$$\begin{aligned} \text{where} \quad \mathcal{R} \ = \ \{(y, d) : y \in \mathcal{H}, d \in \mathcal{D}, \\ y(i, j) = d(i, j) \text{ for all } \ (i, j) \in \mathcal{I}_{\text{first}}\} \end{aligned}$$

This problem has a very similar structure to the problem of integrated parsing and

tagging, and we can derive a similar dual decomposition algorithm. The Lagrange multipliers u are a vector in $\mathcal{R}^{|\mathcal{I}_{\text{first}}|}$ enforcing agreement between dependency assignments. The algorithm (omitted for brevity) is identical to the algorithm in figure 2-1, but with $\mathcal{I}_{\text{uni}}$, $\mathcal{Y}$, $\mathcal{Z}$, θ^{cfg}, and θ^{tag} replaced with $\mathcal{I}_{\text{first}}$, $\mathcal{H}$, $\mathcal{D}$, θ^{head}, and θ^{dep} respectively. The algorithm only requires decoding algorithms for the two models, together with simple updates to the Lagrange multipliers.

2.4 Marginal Polytopes and LP Relaxations

We now give formal guarantees for the algorithms in the previous section, showing that they solve LP relaxations of the problems in Eqs. 2.2 and 2.3.

To make the connection to linear programming, we first introduce the idea of *marginal polytopes* in section 2.4.1. In section 2.4.2, we give a precise statement of the LP relaxations that are being solved by the example algorithms, making direct use of marginal polytopes. In section 2.5 we will prove that the example algorithms solve these LP relaxations.

2.4.1 Marginal Polytopes

For a finite set $\mathcal{Y}$, define the set of all distributions over elements in $\mathcal{Y}$ as $\Delta = \{\alpha \in \mathcal{R}^{|\mathcal{Y}|} : \alpha_y \geq 0, \sum_{y \in \mathcal{Y}} \alpha_y = 1\}$. Each $\alpha \in \Delta$ gives a vector of marginals, $\mu = \sum_{y \in \mathcal{Y}} \alpha_y y$, where μ_r can be interpreted as the probability that $y_r = 1$ for a y selected at random from the distribution α.

The set of all possible marginal vectors, known as the *marginal polytope*, is defined as follows:

$$\mathcal{M} = \{\mu \in \mathcal{R}^m : \exists \alpha \in \Delta \text{ such that } \mu = \sum_{y \in \mathcal{Y}} \alpha_y y\}$$

$\mathcal{M}$ is also frequently referred to as the *convex hull* of $\mathcal{Y}$, written as $\text{conv}(\mathcal{Y})$. We will use the notation $\text{conv}(\mathcal{Y})$ in the remainder of this chapter.

For an arbitrary set $\mathcal{Y}$, the marginal polytope $\text{conv}(\mathcal{Y})$ can be quite complex.[4] However, Martin et al. [1990] show that for a very general class of dynamic programming problems, the implied marginal polytope can be expressed in the form

$$\text{conv}(\mathcal{Y}) = \{\mu \in \mathcal{R}^m : A\mu = b, \mu \geq 0\} \tag{2.4}$$

[4]For any finite set $\mathcal{Y}$, $\text{conv}(\mathcal{Y})$ can be expressed as $\{\mu \in \mathcal{R}^k : A\mu \leq b\}$ where A is a matrix of dimension $p \times k$, and $b \in \mathcal{R}^p$ (e.g., Korte and Vygen [2008], page 65). However the value for p depends on the set $\mathcal{Y}$, and can be exponential in size.

$$\forall r \in \mathcal{I}',\ \mu_r \geq 0\ ; \quad \sum_{\substack{X,Y,Z \in N \\ k=1\ldots(n-1)}} \mu(X \rightarrow Y\ Z, 1, k, n) = 1 \tag{2.5}$$

$\forall X \in N$, $\forall (i,j)$ such that $1 \leq i < j \leq n$ and $(i,j) \neq (1,n)$:

$$\sum_{\substack{Y,Z \in N \\ k=i\ldots(j-1)}} \mu(X \rightarrow Y\ Z, i, k, j) = \sum_{\substack{Y,Z \in N \\ k=1\ldots(i-1)}} \mu(Y \rightarrow Z\ X, k, i-1, j) + \sum_{\substack{Y,Z \in N \\ k=(j+1)\ldots n}} \mu(Y \rightarrow X\ Z, i, j, k) \tag{2.6}$$

$$\forall Y \in T,\ \forall i \in \{1 \ldots n\} : \quad \mu(i, Y) = \sum_{\substack{X,Z \in N \\ k=(i+1)\ldots n}} \mu(X \rightarrow Y\ Z, i, i, k) + \sum_{\substack{X,Z \in N \\ k=1\ldots(i-1)}} \mu(X \rightarrow Z\ Y, k, i-1, i) \tag{2.7}$$

Figure 2-2: The linear constraints defining the marginal polytope for CFG parsing.

where A is a $p \times m$ matrix, b is vector in $\mathcal{R}^p$, and the value p is linear in the size of a hypergraph representation of the dynamic program. Note that A and b specify a set of p linear constraints.

We now give an explicit description of the resulting constraints for CFG parsing:[5] similar constraints arise for other dynamic programming algorithms for parsing, for example the algorithms of Eisner [2000b]. The exact form of the constraints, and the fact that they are polynomial in number, is not essential for the formal results in this chapter. However, a description of the constraints gives valuable intuition for the structure of the marginal polytope.

The constraints are given in figure 5-3. To develop some intuition, consider the case where the variables μ_r are restricted to be binary: hence each binary vector μ specifies a parse tree. The second constraint in Eq. 2.5 specifies that exactly one rule must be used at the top of the tree. The set of constraints in Eq. 2.6 specify that for each production of the form $\langle X \rightarrow Y\ Z, i, k, j \rangle$ in a parse tree, there must be exactly one production higher in the tree that generates (X, i, j) as one of its children. The constraints in Eq. 2.7 enforce consistency between the $\mu(i, Y)$ variables and rule variables higher in the tree. Note that the constraints in Eqs.(2.5–2.7) can be written in the form $A\mu = b$, $\mu \geq 0$, as in Eq. 2.4.

Under these definitions, we have the following:

[5] Taskar et al. [2004] describe the same set of constraints, but without proof of correctness or reference to Martin et al. [1990].

$$\forall r \in \mathcal{I}',\ \nu_r \geq 0\ ;\ \sum_{X,Y,Z \in T} \nu((X,Y) \rightarrow Z, 3) = 1$$

$\forall X \in T$, $\forall i \in \{3 \ldots n-1\}$:

$$\sum_{Y,Z \in T} \nu((Y,Z) \rightarrow X, i) = \sum_{Y,Z \in T} \nu((Y,X) \rightarrow Z, i+1)$$

$\forall X \in T$, $\forall i \in \{3 \ldots n-2\}$:

$$\sum_{Y,Z \in T} \nu((Y,Z) \rightarrow X, i) = \sum_{Y,Z \in T} \nu((X,Y) \rightarrow Z, i+2)$$

$$\forall X \in T, \forall i \in \{3 \ldots n\} :\ \nu(i,X) = \sum_{Y,Z \in T} \nu((Y,Z) \rightarrow X, i)$$

$$\forall X \in T :\quad \nu(1,X) = \sum_{Y,Z \in T} \nu((X,Y) \rightarrow Z, 3)$$

$$\forall X \in T :\quad \nu(2,X) = \sum_{Y,Z \in T} \nu((Y,X) \rightarrow Z, 3)$$

Figure 2-3: The linear constraints defining the marginal polytope for trigram POS tagging.

Theorem 1 *Define $\mathcal{Y}$ to be the set of all CFG parses, as defined in section 2.3. Then*

$$\text{conv}(\mathcal{Y}) = \{\mu \in \mathcal{R}^m : \mu \text{ satisifies Eqs.(2.5-2.7)}\}$$

Proof: This theorem is a special case of Martin et al. [1990], theorem 2.

The marginal polytope for tagging, $\text{conv}(\mathcal{Z})$, can also be expressed using linear constraints as in Eq. 2.4. This follows from well-known results for graphical models [Wainwright and Jordan, 2008], or from the Martin et al. [1990] construction. See figure 2-3 for the full set of constraints.

As a final point, the following theorem gives an important property of marginal polytopes, which we will use at several points in this chapter:

Theorem 2 *(Korte and Vygen [2008], page 66.) For any set $\mathcal{Y} \subseteq \{0,1\}^k$, and for any vector $\theta \in \mathcal{R}^k$,*

$$\max_{y \in \mathcal{Y}} y \cdot \theta = \max_{\mu \in \text{conv}(\mathcal{Y})} \mu \cdot \theta \tag{2.8}$$

The theorem states that for a linear objective function, maximization over a discrete set $\mathcal{Y}$ can be replaced by maximization over the convex hull $\text{conv}(\mathcal{Y})$. The problem $\max_{\mu \in \text{conv}(\mathcal{Y})} \mu \cdot \theta$ is a linear programming problem.

For parsing, this theorem implies that:

1. Weighted CFG parsing can be framed as a linear programming problem, of the form $\max_{\mu \in \text{conv}(\mathcal{Y})} \mu \cdot \theta$, where $\text{conv}(\mathcal{Y})$ is specified by a polynomial number of linear constraints.

2. Conversely, dynamic programming algorithms such as the CKY algorithm can be considered to be oracles that efficiently solve LPs of the form $\max_{\mu \in \text{conv}(\mathcal{Y})} \mu \cdot \theta$.

Similar results apply for the POS tagging case.

2.4.2 Linear Programming Relaxations

We now describe the LP relaxations that are solved by the example algorithms in section 2.3. We begin with the algorithm in figure 2-1.

The original optimization problem was to find $\max_{(y,z) \in \mathcal{Q}} \left(y \cdot \theta^{\text{cfg}} + z \cdot \theta^{\text{tag}} \right)$ (see Eq. 2.2). By theorem 2, this is equivalent to solving

$$\max_{(\mu,\nu) \in \text{conv}(\mathcal{Q})} \left(\mu \cdot \theta^{\text{cfg}} + \nu \cdot \theta^{\text{tag}} \right) \tag{2.9}$$

To approximate this problem, we first define

$$\begin{aligned} \mathcal{Q}' = \{(\mu, \nu) : \mu \in \text{conv}(\mathcal{Y}), \nu \in \text{conv}(\mathcal{Z}), \\ \mu(i,t) = \nu(i,t) \text{ for all } (i,t) \in \mathcal{I}_{\text{uni}}\} \end{aligned}$$

This definition is very similar to the definition of $\mathcal{Q}$, but with $\mathcal{Y}$ and $\mathcal{Z}$ replaced by $\text{conv}(\mathcal{Y})$ and $\text{conv}(\mathcal{Z})$. We then define the following problem:

$$\max_{(\mu,\nu) \in \mathcal{Q}'} \left(\mu \cdot \theta^{\text{cfg}} + \nu \cdot \theta^{\text{tag}} \right) \tag{2.10}$$

$\mathcal{Q}'$ is a set that is defined by a finite set of linear constraints; hence this problem is a linear program. Eq. 2.10 is a *relaxation* of the problem in Eq. 2.9, in that we have replaced $\text{conv}(\mathcal{Q})$ with $\mathcal{Q}'$, and $\text{conv}(\mathcal{Q})$ is a subset of $\mathcal{Q}'$ ($\mathcal{Q}'$ is an *outer bound* for $\text{conv}(\mathcal{Q})$). (To see this, note that any point in $\mathcal{Q}$ is clearly in $\mathcal{Q}'$. It follows that any point in $\text{conv}(\mathcal{Q})$ is also in $\mathcal{Q}'$, because $\mathcal{Q}'$ is a convex set.[6]) Appendix A.1 gives an example showing that $\text{conv}(\mathcal{Q})$ is usually a strict subset of $\mathcal{Q}'$, that is, $\mathcal{Q}'$ includes points that are not in $\text{conv}(\mathcal{Q})$.

LP relaxations based on outer bounds of marginal polytopes have been applied in many

[6]This is easily verified, as $\mathcal{Q}'$ is defined by linear constraints.

papers on approximate inference in graphical models, with some success. In general, the solution to Eq. 2.10 may be in $\mathcal{Q}'$ but not in $\text{conv}(\mathcal{Q})$, in which case it will be fractional. However in several empirical settings—including the experiments in this chapter—the relaxation in Eq. 2.10 turns out to be tight, in that the solution is often integral (i.e., it is in $\mathcal{Q}$). In these cases solving the relaxed LP *exactly* solves the original problem of interest.

In the next section we prove that the algorithm in figure 2-1 solves the problem in Eq 2.10. A similar result holds for the algorithm in section 2.3.2: it solves a relaxation of Eq. 2.3, where $\mathcal{R}$ is replaced by

$$\begin{aligned}\mathcal{R}' = \{(\mu, \nu) : \mu \in \text{conv}(\mathcal{H}), \nu \in \text{conv}(\mathcal{D}),\\ \mu(i,j) = \nu(i,j) \text{ for all } (i,j) \in \mathcal{I}_{\text{first}}\}\end{aligned}$$

2.5 Convergence Guarantees

2.5.1 Lagrangian Relaxation

We now show that the example algorithms solve their respective LP relaxations given in the previous section. We do this by first introducing a general class of linear programs, together with an optimization method, Lagrangian relaxation, for solving these linear programs. We then show that the algorithms in section 2.3 are special cases of the general algorithm.

The linear programs we consider take the form

$$\max_{x_1 \in X_1, x_2 \in X_2} (\theta_1 \cdot x_1 + \theta_2 \cdot x_2) \quad \text{such that } Ex_1 = Fx_2$$

The matrices $E \in \mathcal{R}^{q \times m}$ and $F \in \mathcal{R}^{q \times l}$ specify q linear "agreement" constraints between $x_1 \in \mathcal{R}^m$ and $x_2 \in \mathcal{R}^l$. The sets X_1, X_2 are also specified by linear constraints, $X_1 = \{x_1 \in \mathcal{R}^m : Ax_1 = b, x_1 \geq 0\}$ and $X_2 = \{x_2 \in \mathcal{R}^l : Cx_2 = d, x_2 \geq 0\}$, hence the problem is an LP.

It is natural to apply Lagrangian relaxation in cases where the sub-problems $\max_{x_1 \in X_1} \theta_1 \cdot x_1$ and $\max_{x_2 \in X_2} \theta_2 \cdot x_2$ can be efficiently solved by combinatorial algorithms for any values of θ_1, θ_2, but where the constraints $Ex_1 = Fx_2$ "complicate" the problem. We introduce

Lagrange multipliers $u \in \mathcal{R}^q$ that enforce the latter set of constraints, giving the Lagrangian:

$$L(u, x_1, x_2) = \theta_1 \cdot x_1 + \theta_2 \cdot x_2 + u \cdot (Ex_1 - Fx_2)$$

The dual objective function is

$$L(u) = \max_{x_1 \in X_1, x_2 \in X_2} L(u, x_1, x_2)$$

and the dual problem is to find $\min_{u \in \mathcal{R}^q} L(u)$.

Because X_1 and X_2 are defined by linear constraints, by strong duality we have

$$\min_{u \in \mathcal{R}^q} L(u) = \max_{x_1 \in X_1, x_2 \in X_2 : Ex_1 = Fx_2} (\theta_1 \cdot x_1 + \theta_2 \cdot x_2)$$

Hence minimizing $L(u)$ will recover the maximum value of the original problem. This leaves open the question of how to recover the LP solution (i.e., the pair (x_1^*, x_2^*) that achieves this maximum); we discuss this point in section 2.5.2.

The dual $L(u)$ is convex. However, $L(u)$ is not differentiable, so we cannot use gradient-based methods to optimize it. Instead, a standard approach is to use a subgradient method. Subgradients are tangent lines that lower bound a function even at points of non-differentiability: formally, a subgradient of a convex function $L : \mathcal{R}^n \to \mathcal{R}$ at a point u is a vector g_u such that for all v, $L(v) \geq L(u) + g_u \cdot (v - u)$. By standard results, the subgradient for L at a point u takes a simple form, $g_u = Ex_1^* - Fx_2^*$, where

$$x_1^* = \arg\max_{x_1 \in X_1} (\theta_1 + (u^{(k)})^T E) \cdot x_1$$

$$x_2^* = \arg\max_{x_2 \in X_2} (\theta_2 - (u^{(k)})^T F) \cdot x_2$$

The beauty of this result is that the values of x_1^* and x_2^*, and by implication the value of the subgradient, can be computed using oracles for the two arg max sub-problems.

Subgradient algorithms perform updates that are similar to gradient descent:

$$u^{(k+1)} \leftarrow u^{(k)} - \alpha_k g^{(k)}$$

where $g^{(k)}$ is the subgradient of L at $u^{(k)}$ and $\alpha_k > 0$ is the step size of the update. The complete sub-gradient algorithm is given in figure 2-4. The following convergence theorem

$u^{(1)} \leftarrow 0$
for $k = 1$ to K **do**
  $x_1^{(k)} \leftarrow \arg\max_{x_1 \in X_1} (\theta_1 + (u^{(k)})^T E) \cdot x_1$
  $x_2^{(k)} \leftarrow \arg\max_{x_2 \in X_2} (\theta_2 - (u^{(k)})^T F) \cdot x_2$
  if $Ex_1^{(k)} = Fx_2^{(k)}$ **return** $u^{(k)}$
  $u^{(k+1)} \leftarrow u^{(k)} - \alpha_k (Ex_1^{(k)} - Fx_2^{(k)})$
return $u^{(K)}$

Figure 2-4: The Lagrangian relaxation algorithm.

is well-known (e.g., see page 120 of Korte and Vygen [2008]):

Theorem 3 *If* $\lim_{k\to\infty} \alpha_k = 0$ *and* $\sum_{k=1}^{\infty} \alpha_k = \infty$, *then* $\lim_{k\to\infty} L(u^{(k)}) = \min_u L(u)$.

The following proposition is easily verified:

Proposition 1 *The algorithm in figure 2-1 is an instantiation of the algorithm in figure 2-4,*[7] *with* $X_1 = \text{conv}(\mathcal{Y})$, $X_2 = \text{conv}(\mathcal{Z})$, *and the matrices* E *and* F *defined to be binary matrices specifying the constraints* $y(i,t) = z(i,t)$ *for all* $(i,t) \in I_{uni}$.

Under an appropriate definition of the step sizes α_k, it follows that the algorithm in figure 2-1 defines a sequence of Lagrange multipliers $u^{(k)}$ minimizing a dual of the LP relaxation in Eq. 2.10. A similar result holds for the algorithm in section 2.3.2.

2.5.2 Recovering the LP Solution

The previous section described how the method in figure 2-4 can be used to minimize the dual $L(u)$ of the original linear program. We now turn to the problem of recovering a primal solution (x_1^*, x_2^*) of the LP. The method we propose considers two cases:

(Case 1) If $Ex_1^{(k)} = Fx_2^{(k)}$ at any stage during the algorithm, then simply take $(x_1^{(k)}, x_2^{(k)})$ to be the primal solution. In this case the pair $(x_1^{(k)}, x_2^{(k)})$ *exactly* solves the original LP.[8] If this case arises in the algorithm in figure 2-1, then the resulting solution is binary (i.e., it is a member of $\mathcal{Q}$), and the solution exactly solves the original inference problem.

[7] with the caveat that it returns $(x_1^{(k)}, x_2^{(k)})$ rather than $u^{(k)}$.

[8] We have that $\theta_1 \cdot x_1^{(k)} + \theta_2 \cdot x_2^{(k)} = L(u^{(k)}, x_1^{(k)}, x_2^{(k)}) = L(u^{(k)})$, where the last equality is because $x_1^{(k)}$ and $x_2^{(k)}$ are defined by the respective arg max's. Thus, $(x_1^{(k)}, x_2^{(k)})$ and $u^{(k)}$ are primal and dual optimal.

(Case 2) If case 1 does not arise, then a couple of strategies are possible. (This situation could arise in cases where the LP is not tight—i.e., it has a fractional solution—or where K is not large enough for convergence.) The first is to define the primal solution to be the average of the solutions encountered during the algorithm: $\hat{x}_1 = \sum_k x_1^{(k)}/K$, $\hat{x}_2 = \sum_k x_2^{(k)}/K$. Results from Nedić and Ozdaglar [2009] show that as $K \to \infty$, these averaged solutions converge to the optimal primal solution.[9] A second strategy (as given in figure 2-1) is to simply take $(x_1^{(K)}, x_2^{(K)})$ as an approximation to the primal solution. This method is a heuristic, but previous work (e.g., Komodakis et al. [2007]) has shown that it is effective in practice; we use it in this chapter.

In our experiments we found that in the vast majority of cases, case 1 applies, after a relatively small number of iterations of the algorithm: see the next section for more details.

2.6 Experiments

2.6.1 Integrated Phrase-Structure and Dependency Parsing

Our first set of experiments considers the integration of Model 1 of Collins [2003] (a lexicalized phrase-structure parser, from here on referred to as Model 1),[10] and the 2nd order discriminative dependency parser of Koo et al. [2008]. The inference problem for a sentence x is to find

$$y^* = \arg\max_{y \in \mathcal{Y}} (f_1(y) + \gamma f_2(y)) \tag{2.11}$$

where $\mathcal{Y}$ is the set of all lexicalized phrase-structure trees for the sentence x; $f_1(y)$ is the score (log probability) under Model 1; $f_2(y)$ is the score under Koo et al. [2008] for the dependency structure implied by y; and $\gamma > 0$ is a parameter dictating the relative weight of the two models. This problem is similar to the second example in section 2.3; a very similar dual decomposition algorithm to that described in section 2.3.2 can be derived.

We used the Penn Wall Street Treebank [Marcus et al., 1994] for the experiments, with sections 2-21 for training, section 22 for development, and section 23 for testing. The parameter γ was chosen to optimize performance on the development set.

We ran the dual decomposition algorithm with a limit of $K = 50$ iterations. Note again

[9] The resulting fractional solution can be projected back to the set $\mathcal{Q}$, see [Smith and Eisner, 2008b, Martins et al., 2009b].

[10] We use a reimplementation that is a slight modification of Collins Model 1, with very similar performance, and which uses the TAG formalism of Carreras et al. [2008].

Itn.	1	2	3	4	5-10	11-20	20-50	**
Dep	43.5	20.1	10.2	4.9	14.0	5.7	1.4	0.4
POS	58.7	15.4	6.3	3.6	10.3	3.8	0.8	1.1

Table 2.1: Convergence results for Section 23 of the WSJ Treebank for the dependency parsing and POS experiments. Each column gives the percentage of sentences whose *exact* solutions were found in a given range of subgradient iterations. ** is the percentage of sentences that did not converge by the iteration limit (K=50).

	Precision	Recall	F_1	Dep
Model 1	88.4	87.8	88.1	91.4
Koo08 Baseline	89.9	89.6	89.7	93.3
DD Combination	91.0	90.4	90.7	93.8

Table 2.2: Performance results for Section 23 of the WSJ Treebank. Model 1: a reimplementation of the generative parser of [Collins, 2002b]. Koo08 Baseline: Model 1 with a hard restriction to dependencies predicted by the discriminative dependency parser of [Koo et al., 2008]. DD Combination: a model that maximizes the joint score of the two parsers. Dep shows the unlabeled dependency accuracy of each system.

that the dual decomposition algorithm returns an exact solution if case 1 occurs as defined in section 2.5.2; we found that of 2416 sentences in section 23, case 1 occurred for 2407 (99.6%) sentences. Table 2.1 gives statistics showing the number of iterations required for convergence. Over 80% of the examples converge in 5 iterations or fewer; over 90% converge in 10 iterations or fewer.

We compare the accuracy of the dual decomposition approach to two baselines: first, Model 1; and second, a naive integration method that enforces the hard constraint that Model 1 must only consider dependencies seen in the first-best output from the dependency parser. Table 2.2 shows all three results. The dual decomposition method gives a significant gain in precision and recall over the naive combination method, and boosts the performance of Model 1 to a level that is close to some of the best single-pass parsers on the Penn treebank test set. Dependency accuracy is also improved over the Koo et al. [2008] model, in spite of the relatively low dependency accuracy of Model 1 alone.

Figure 4-4 shows performance of the approach as a function of K, the maximum number of iterations of dual decomposition. For this experiment, for cases where the method has not converged for $k \leq K$, the output from the algorithm is chosen to be the $y^{(k)}$ for $k \leq K$ that maximizes the objective function in Eq. 2.11. The graphs show that values of K less than 50 produce almost identical performance to $K = 50$, but with fewer cases giving certificates of optimality (with $K = 10$, the f-score of the method is 90.69%; with $K = 5$ it is 90.63%).

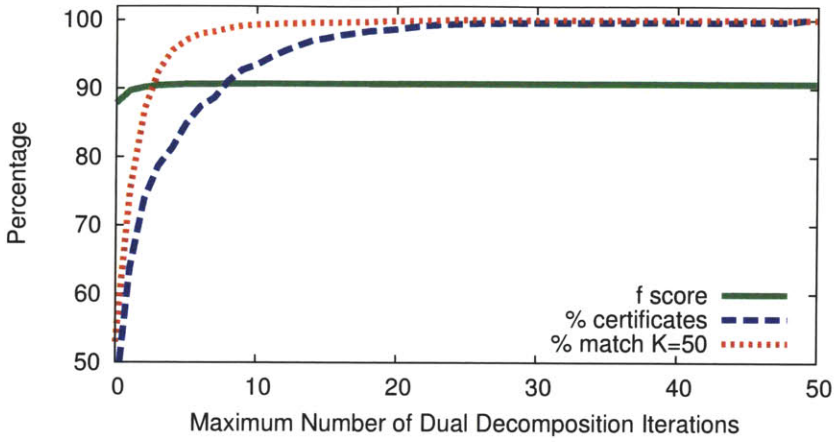


Figure 2-5: Performance on the parsing task assuming a fixed number of iterations K. f-score: accuracy of the method. % certificates: percentage of examples for which a certificate of optimality is provided. % match: percentage of cases where the output from the method is identical to the output when using $K = 50$.

	Precision	Recall	F_1	POS Acc
Fixed Tags	88.1	87.6	87.9	96.7
DD Combination	88.7	88.0	88.3	97.1

Table 2.3: Performance results for Section 23 of the WSJ. Model 1 (Fixed Tags): a baseline parser initialized to the best tag sequence of from the tagger of Toutanova and Manning [2000]. DD Combination: a model that maximizes the joint score of parse and tag selection.

2.6.2 Integrated Phrase-Structure Parsing and Trigram POS tagging

In a second experiment, we used dual decomposition to integrate the Model 1 parser with the Stanford max-ent trigram POS tagger [Toutanova and Manning, 2000], using a very similar algorithm to that described in section 2.3.1. We use the same training/dev/test split as in section 2.6.1.

We ran the algorithm with a limit of $K = 50$ iterations. Out of 2416 test examples, the algorithm found an exact solution in 98.9% of the cases. Table 2.1 gives statistics showing the speed of convergence for different examples: over 94% of the examples converge to an exact solution in 10 iterations or fewer. In terms of accuracy, we compare to a baseline approach of using the first-best tag sequence as input to the parser. The dual decomposition approach gives 88.3 F1 measure in recovering parse-tree constituents, compared to 87.9 for the baseline.

2.7 Conclusions

We have introduced dual-decomposition algorithms for inference in NLP, given formal properties of the algorithms in terms of LP relaxations, and demonstrated their effectiveness on problems that would traditionally be solved using intersections of dynamic programs (e.g., Bar-Hillel et al. [1964]). Given the widespread use of dynamic programming in NLP, there should be many applications for the approach.

There are several possible extensions of the method we have described. We have focused on cases where two models are being combined; the extension to more than two models is straightforward (e.g., see [Komodakis et al., 2007]). This paper has considered approaches for MAP inference; for closely related methods that compute approximate marginals see [Wainwright et al., 2005b].

Chapter 3

Incorporating Inter-Sentence Constraints

This chapter is joint work with Roi Reichart, Michael Collins, and Amir Globerson.

3.1 Introduction

In this chapter we address a different decoding challenge: decoding with a global objective. Instead of combining two overlapping models in a single sentence, we integrate many sentence-level models with a series of inter-sentence constraints. This global information can improve model accuracy, particularly when we have access to limited training data. Even though each sentence-level model can be solved efficiently, the constraints tie together the entire corpus, which makes exact decoding intractable.

We consider examples of this problem in part-of-speech (POS) tagging and dependency parsing. In POS tagging, most taggers perform very well on word types that they have observed in training data, but they perform poorly on unknown words. With a global objective, we can include constraints that encourage a consistent tag across all occurrences of an unknown word type to improve accuracy. In dependency parsing, the parser can benefit from surface-level features of the sentence, but with sparse training data these features are very noisy. Using a global objective, we can add constraints that encourage similar surface-level contexts to exhibit similar syntactic behaviour.

We utilize Markov random fields (MRFs) to enforce global constraints between sen-

tences. We represent each word as a node, the tagging or parse decision as its label, and add constraints through edges. MRFs allow us to include global constraints tailored to dependency parsing and tagging, and to reason about inference in the corresponding global models.

The contribution of this chapter is a dual decomposition algorithm for decoding a global objective with inter-sentence constraints. Our algorithm splits the global inference problem into subproblems - sentence-level problems and the global MRF. These subproblems can be solved efficiently through known methods. We show empirically that by iteratively solving these subproblems, we can find the exact solution to the global model.

In experiments, we demonstrate that global models with inter-sentence constraints can improve upon state-of-the-art sentence-level models for dependency parsing and part-of-speech tagging in the scenario of sparse training data. For dependency parsing, we show an absolute gain of 2.5% over the second-order projective MST parser [McDonald et al., 2005b]. For POS tagging, we show an absolute gain of 2.2% over the Stanford trigram tagger [Toutanova et al., 2003]. In both tasks, our algorithm finds the exact solution to the global corpus-level objective over 95% of the time and for most experiments requires less than 100 efficient iterations.

3.2 Background: Structured Models

We begin by introducing notation for the sentence-level models that form the basis for this work. We focus on structured models for two important problems in natural language processing, dependency parsing and part-of-speech tagging. We also introduce notation for Markov random fields, the framework we use for enforcing constraints.

The first structured model we consider is for dependency parsing. The goal is to find the best parse y for a tagged sentence $x = (w_1/t_1, \ldots, w_n/t_n)$ with words w and POS tags t. Define the *index set* for dependency parsing as,

$$\mathcal{I}(x) = \{(m, h) : m \in \{1 \ldots n\}, \\ h \in \{0 \ldots n\}, m \neq h\}$$

where $h = 0$ represents the root word. A dependency parse is a vector $y = \{y(m, h) : (m, h) \in \mathcal{I}(x)\}$ where $y(m, h) = 1$ if m is a modifier of the head word h. We define the

set $\mathcal{Y}(x) \subset \{0,1\}^{|\mathcal{I}(x)|}$ to be the set of all valid dependency parses for a sentence x. In this work, we use projective dependency parses, but the method applies analogously to the set of non-projective parse trees.

Additionally, we have a scoring function $f : \mathcal{Y}(x) \rightarrow \mathcal{R}$. The optimal parse y^* for a sentence x is given by, $y^* = \arg\max_{y \in \mathcal{Y}(x)} f(y)$. This local decoding problem can often be solved efficiently. For example in commonly used projective dependency parsing models, we can compute y^* efficiently using variants of the Viterbi algorithm.

For this work, we make the assumption that we have an efficient algorithm to find the argmax of

$$f(y) + \sum_{(m,h) \in \mathcal{I}(x)} u(m,h)y(m,h) = f(y) + u \cdot y$$

where u is a vector in $\mathcal{R}^{|\mathcal{I}(x)|}$. In practice, u will be a vector of Lagrange multipliers associated with the dependencies of y in our dual decomposition algorithm.

We can construct a very similar setting for POS tagging. The goal is to find the best tagging y for a sentence $x = (w_1, \ldots, w_n)$. The index set for POS tagging is

$$\mathcal{I}^{\mathrm{T}}(x) = \{(i,t) : i \in \{1 \ldots n\}, t \in \mathcal{T}\}$$

where $\mathcal{T}$ is the set of possible tags. A tagging is a vector $y = \{y(i,t) : (i,t) \in \mathcal{I}^{\mathrm{T}}\}$ where $y(i,t) = 1$ if word i has tag t. We define the set $\mathcal{Y}^{\mathrm{T}}(x) \subset \{0,1\}^{|\mathcal{I}^{\mathrm{T}}(x)|}$ to be the set of all valid taggings for a sentence x.

We additionally have a scoring function over taggings $f^{\mathrm{T}} : \mathcal{Y}^{\mathrm{T}}(x) \rightarrow \mathcal{R}$. The optimal tagging y^* for a sentence x is given by: $y^* = \arg\max_{y \in \mathcal{Y}^{\mathrm{T}}(x)} f^{\mathrm{T}}(y)$. In commonly used tagging models, for instance trigram taggers, this problem can be solved efficiently using the Viterbi algorithm. We also assume an efficient algorithm to find the argmax of

$$f^{\mathrm{T}}(y) + \sum_{(i,t) \in \mathcal{I}^{\mathrm{T}}(x)} u(i,t)y(i,t) = f^{\mathrm{T}}(y) + u \cdot y$$

where u is a vector in $\mathcal{R}^{|\mathcal{I}^{\mathrm{T}}(x)|}$. The vector u will be a vector of Lagrange multipliers associated with the tags of y in our dual decomposition algorithm.

We next introduce Markov random fields (MRFs) which will give us a convenient framework for specifying global constraints. An MRF consists of an undirected graph $G = (V, E)$,

a set of possible labels for each node $L = \{L_1, \ldots, L_{|V|}\}$, and a scoring function g. The index set for MRFs is

$$\mathcal{I}^{\text{MRF}} = \{((i,j), l_i, l_j) : (i,j) \in E \text{ and } l_i \in L_i, l_j \in L_j\}$$

where the l_i and l_j are the labels chosen for adjacent nodes i and j. A label assignment in the MRF is a binary vector z with $z((i,j), l_i, l_j) = 1$ if the labels l_i, l_j are selected for the nodes i, j. The set of all valid label assignments (one label per node) is given by $\mathcal{Z} \subset \{0,1\}^{|\mathcal{I}^{\text{MRF}}|}$.

The best assignment z^* in an MRF is determined by a scoring function g,

$$z^* = \arg\max_{z \in \mathcal{Z}} g(z)$$

where $g : \mathcal{Z} \to \mathcal{R}$. For simplicity, we focus on the subclass of *pairwise* MRFs where this function g is a linear function of z,

$$\begin{aligned} g(z) &= \sum_{((i,j),l_i,l_j) \in \mathcal{I}^{\text{MRF}}} z((i,j), l_i, l_j)\theta((i,j), l_i, l_j) \\ &= z \cdot \theta \end{aligned}$$

where θ is known as the vector of edge potentials. The optimization problem for MRFs can be solved with the junction tree algorithm and has complexity exponential in the tree width of the graph G.

For convenience, we extend the index set $\mathcal{I}$ to

$$\mathcal{I}'^{\text{MRF}} = \mathcal{I}^{\text{MRF}} \cup \{(\nu, l) : \nu \in \{1 \ldots |V|\}, l \in L_\nu\}$$

Under the extended index set $z(\nu, l) = 1$ when node ν has label l. We note that we can always incorporate an additional set of weights into the objective,

$$g(z) + \sum_{(\nu,l) \in \mathcal{I}^{\text{MRF}}(x)} u(\nu, l) z(\nu, l) = z \cdot \theta'$$

i.e. additional node potentials can simply be incorporated into the vector of edge potentials without changing the structure of the scoring function.

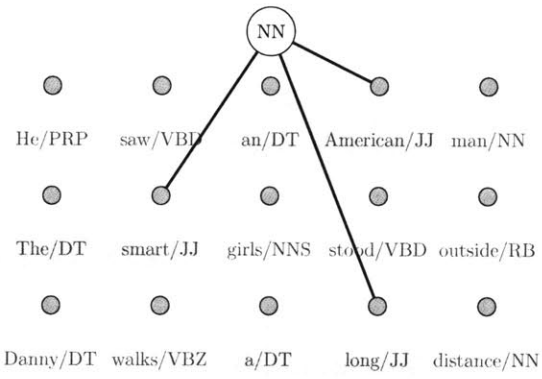


Figure 3-1: An example constraint from dependency parsing. Each gray node corresponds to a possible modifier in the test corpus. The constraint applies to all modifiers in the context `DT JJ`. The white node corresponds to the consensus POS tag of the head word of these modifiers.

3.3 A Parsing Example

In this section we give a detailed example of global constraints for dependency parsing. The aim is to construct a global objective that encourages similar contexts across the corpus to exhibit similar syntactic behaviour. We implement this constraint using an MRF with a node for each word in the corpus. The label of each node is the index of the word it modifies. We add edges to this MRF to reward consistency among similar contexts.

More specifically, we say that the context of a word is its POS tag and the POS tags of some set of the words around it. We expand on this notion of context in section 3.7, for simplicity we assume here that the context includes only the previous word's POS tag. Our constraints are designed to encourage words in the same context to modify words with similar POS tags. We enforce this property by connecting words that appear in the same context to a shared consensus node. At inference time, the consensus node can take any POS tag as its label. The scoring function encourages word nodes to choose a head word with a similar POS tag to the label of the consensus node.

Figure 3-1 gives an example of a global MRF over a small corpus. The MRF contains a node associated with each word instance in the corpus, where the label of the node is the index of the word it modifies. In this corpus, the context `DT JJ` appears three times. We hope to choose head words with similar POS tags for each one of these three contexts. We enforce this constraint by adding a consensus node and connecting it to the nodes of these words. The MRF scoring function encourages each word in the context to select a head

with the POS chosen for the consensus node.

More concretely, consider the following possible label assignment and parse selection. Assign the label `NN` to the consensus node, and let the word `American` modify the word `saw` which assigns the label `2` to node `(1,4)` (sentence 1, index 4). Since the word `saw` has the POS tag `VBD`, which does not match the consensus node, this assignment has a low MRF score. To increase the total model score, we need to either change the consensus label or modify the parse structure. In this example, the other words in the context `DT JJ`, `smart` and `long`, strongly prefer to modify words with POS tags similar to `NN`. Therefore the best choice is to modify the parse structure of the first sentence to agree with the consensus node.

We construct a similar model for POS tagging. The global MRF includes a node for each word in the corpus where the labels correspond to the POS tag selected for that word. Instead of using contexts, we enforce similarity among word instances of the same word type. We include a consensus node for each word type, and connect this node to all instances of this type. The edges encourage the word nodes to choose the same label as the consensus node, and thus enforce global consistency among the POS tags chosen for the same word type.

3.4 Global Objective

We now give a formal description of models with inter-sentence constraints. Recall the definition of sentence-level parsing, where the optimal parse y^* for a single sentence x under a scoring function f is given by: $y^* = \arg\max_{y \in \mathcal{Y}(x)} f(y)$. In practice, it is common to apply this objective to an entire corpus of sentences. We can make this formal by giving a simple objective for scoring an entire corpus, specified by the tuple of sentences $X = (x_1, ..., x_r)$, and the product of possible parses $\mathcal{Y}(X) = \mathcal{Y}(x_1) \times \ldots \times \mathcal{Y}(x_r)$. The sentence-level decoding goal is to find the optimal corresponding dependency parses $Y^* = (Y_1^*, ..., Y_r^*) \in \mathcal{Y}(X)$ under a global objective:

$$Y^* = \arg \max_{Y \in \mathcal{Y}(X)} F(Y) = \arg \max_{Y \in \mathcal{Y}(X)} \sum_{s=1}^{r} f(Y_s)$$

where $F : \mathcal{Y}(X) \rightarrow \mathcal{R}$ is the global scoring function. It is easy to see that this objective factors back into independent sentence-level problems:

$$Y^* = (\arg \max_{y \in \mathcal{Y}(x_1)} f(y), \ldots, \arg \max_{y \in \mathcal{Y}(x_r)} f(y))$$

We now consider scoring functions where the global objective includes inter-sentence constraints. Objectives of this form will not factor directly into individual parsing problems; however, we can choose to write them as the sum of two convenient terms:

- A simple sum of sentence-level objectives.
- A global MRF that connects the local structures.

Recall that an MRF is specified by a graph $G = (V, E)$, label sets $L_1, \ldots, L_{|V|}$, and scoring function g. For each sentence in the document $x_s = (w_1/t_1, \ldots, w_n/t_n)$, our MRF includes nodes $\{(s, 1), \ldots, (s, n)\} \subset V$, corresponding to each word in the corpus, and label sets $L_{(s,i)} = \{0, \ldots, n\}$, corresponding to possible head words. Additionally, we may have other nodes in the MRF to enforce specific constraints, for instance the consensus node given in section 3.3. As before, the set $\mathcal{Z}$ includes all valid label assignments in this MRF.

For convenience, we define the index set,

$$\mathcal{J}(X) = \{(s, m, h) : s \in \{1, \ldots, r\}, (m, h) \in \mathcal{I}(x_s)\}$$

which enumerates all possible dependencies at each sentence in the corpus. We say the parses Y_s are consistent with a label assignment z if for all $(s, m, h) \in \mathcal{J}(X)$, $z((s, m), h) = Y_s(m, h)$, i.e. that labels in z match the head words chosen in parse Y_s.

The full global decoding objective is,

$$(Y^*, z^*) = \arg \max_{Y \in \mathcal{Y}(X),\, z \in \mathcal{Z}} F(Y) + g(z)$$

$$\text{s.t. } \forall (s, m, h) \in \mathcal{J}(X),\ z((s, m), h) = Y_s(m, h)$$

The solution to this objective maximizes the local models as well as the global MRF, while maintaining consistency among the models.

The global objective for POS tagging has a similar form. As before we add a node to the MRF for each word in the corpus. We use the POS tag set as our labels for each of

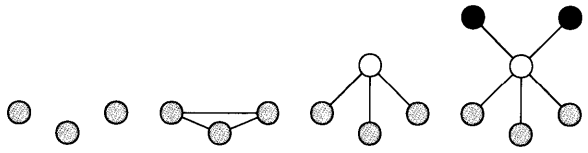


Figure 3-2: Example MRF topologies . Black nodes are constrained to take a fixed label, gray nodes are constrained to take on the label chosen by the local model, and white nodes are unconstrained.

these nodes. We then define the index set

$$\mathcal{J}^{\mathrm{T}}(X) = \{(s,i,t) : s \in \{1,\ldots,r\}, \\ (i,t) \in \mathcal{I}^{\mathrm{T}}(x_s)\}$$

which contains an element for each possible tag at each word instance in the corpus.

The full POS decoding objective is,

$$(Y^*, z^*) = \arg \max_{Y \in \mathcal{Y}(X),\, z \in \mathcal{Z}} F(Y) + g(z)$$
$$\text{s.t. } \forall (s,i,t) \in \mathcal{J}^{\mathrm{T}}(X),\ z((s,i),t) = Y_s(i,t)$$

3.5 MRF Structure

In this section we explore the design decisions in selecting the global MRF. The two important considerations are the topological structure of G, which determines the independence assumptions made in the model, and the scoring function g, which determines the relative strength of the global constraints. These choices will also help in reasoning about complexity of inference in models with global constraints.

We begin by exploring the graph topology of G using dependency parsing as a running example. The most basic topology of G is a single unconnected node for each word token in the corpus. This topology implies that each dependency decision is made independently from the decisions made for other word in the corpus. Figure 3-2(a) gives a graphical representation of this structure.

On the other extreme, we may choose to fully connect a subset of nodes. For instance, in dependency parsing we connect all words that appear in a similar context. Under this

topology, every word is dependent on similar instances in the corpus. Figure 3-2(b) shows the fully-connected topology.

A simpler design is to make a naive Bayes assumption and assume that each node in the set is biased to agree with a shared consensus node. In this graph, decisions at each node are made independently of decisions elsewhere in the corpus given the consensus node. Figure 3-2(c) gives the naive Bayes topology.

Finally, we may choose to add additional nodes that bring in external sources of information, for instance, nodes for the dependency decisions observed in the training data. These nodes can help influence the model, and since they have a fixed observed value, they do not affect the efficiency of inference. Figure 3-2(d) extends the naive Bayes topology to include additional fixed nodes.

In addition to the graph topology, we need to choose a scoring function g over the label assignments in the MRF. Recall that for pairwise MRFs, the scoring function can be represented as a vector of potentials θ specifying the score associated with the labels of two connected nodes. We use these potentials to enforce similarity among inter-sentence nodes in the model.

Consider the scoring function for POS tagging with the naive Bayes assumption as an example. In this model, the word nodes and the consensus node use the tags $\mathcal{T}$ as their label set. To enforce a hard constraint that all connected nodes have the same POS tag, we define θ as

$$\theta_{hard}((i,j),l_i,l_j) = \begin{cases} 0 & \text{if } l_i = l_j \\ -\infty & \text{otherwise} \end{cases}$$

We can also enforce a soft version of this constraint that encourages nodes to agree, but allows some deviation. For label set with more than two labels, an MRF with this scoring function is known as a Potts model and is given by

$$\theta_{potts}((i,j),l_i,l_j) = \begin{cases} \gamma & \text{if } l_i = l_j \\ 0 & \text{otherwise} \end{cases}$$

A further refinement is to scale this penalty to encourage close assignments. In our model, we consider two tags to be close if they share a common prefix, e.g. `NN`, `NNP`, `NNPS` are all nouns and are therefore considered to be close. We refer to this function as a similarity-based Potts scoring function and write it as,

$$\theta_{spotts}((i,j),l_i,l_j) = \begin{cases} \gamma & \text{if } l_i = l_j \\ \eta & \text{if } l_i \text{ is close to } l_j \\ 0 & \text{otherwise} \end{cases}$$

The scoring function for dependency parsing has a very similar form. The only additional complication is that the word nodes take labels from the set $\{0,\ldots,n\}$ representing the word they modify, while the consensus nodes take labels from the set $\mathcal{T}$ of POS tags. We use the notation $t(\nu,h)$ to specify the mapping from head indices to POS tags. We can then write the similarity based Potts scoring function for dependency parsing with consensus nodes as,

$$\theta_{spotts}((i,j),l_i,l_j) = \begin{cases} \gamma & \text{if } t(i,l_i) = l_j \\ \eta & \text{if } t(i,l_i) \text{ is close to } l_j \\ 0 & \text{otherwise} \end{cases}$$

where we assume i is a word node and j is a consensus node.

3.6 A Global Decoding Algorithm

We now consider the decoding question: how to find the structure Y^* that maximizes the global objective. We aim for an efficient solution that makes use of the individual solvers at the sentence-level . For this work, we make the assumption that the graph chosen for the MRF is not "loopy", i.e. it has small tree-width, and can therefore be solved efficiently using dynamic programming.

Before we describe our dual decomposition algorithm, we consider the difficulty of solving the global objective directly. We have an efficient dynamic programming algorithm for solving dependency parsing at the sentence-level, and efficient algorithms for solving the MRF. It follows that we could construct an intersected dynamic programming algorithm that maintains the product of states over both models. This algorithm is exact, but it is very inefficient. Solving the intersected dynamic program requires decoding simultaneously over the entire corpus, with an additional multiplicative factor for solving the MRF. On top of this cost, we need to alter the internal structure of the sentence-level models.

In contrast, we can construct a dual decomposition algorithm which is efficient, produces a certificate when it finds an exact solution, and directly uses the sentence-level parsing

models. Consider again the objective for global dependency parsing,

$$(Y^*, z^*) = \arg \max_{Y \in \mathcal{Y}(X),\, z \in \mathcal{Z}} F(Y) + g(z) \tag{3.1}$$
$$\text{s.t. } \forall (s, m, h) \in \mathcal{J}(X),\ z((s, m), h) = Y_s(m, h)$$

We note that the difficulty in decoding this objective comes entirely from the constraints $z((s,m),h) = Y_s(m,h)$. If these were not there, the problem would factor into two parts, an optimization of F over the test corpus $\mathcal{Y}(X)$ and an optimization of g over possible MRF assignments $\mathcal{Z}$. The first optimization problem factors naturally into sentence-level parsing problems and the second can be solved efficiently given our assumptions on the MRF topology G.

Our algorithm avoids the complications introduced by the constraints $z((s,m),h) = Y_s(m,h)$ by introducing variables $u(s,m,h)$ that help to encourage agreement between the local models and the global MRF. We iteratively solve the subproblems over $\mathcal{Y}(X)$ and $\mathcal{Z}$ and adjust the values of $u(s,m,h)$ to help encourage agreement. These values u are actually Lagrangian multipliers for a relaxed global objective.

The full algorithm is given in figure 3-3. We start with the values of u initialized to 0. At each iteration k, we find the best set of parses $Y^{(k)}$ over the entire corpus and the best MRF assignment $z^{(k)}$. We then update the value of u based on the difference between $Y^{(k)}$ and $z^{(k)}$ and a rate parameter α. On the next iteration, we solve the same decoding problems modified by the new value of u. If at any point the current solutions $Y^{(k)}$ and $z^{(k)}$ satisfy the consistency constraint, we return there current values. Otherwise, we stop at a max iteration K and return the values from the last iteration.

We now give a theorem for the formal guarantees of this algorithm.

Theorem 4 *If for some $k \in \{1 \dots K\}$ in the algorithm in Figure 3-3, $Y_s^{(k)}(m,h) = z^{(k)}(s,m,h)$ for all $(s,m,h) \in \mathcal{J}$, then $(Y^{(k)}, z^{(k)})$ is a solution to the maximization problem in Equation 3.1.*

We omit the proof for brevity. It is slight variation of the proof given by Rush et al. [2010].

Set $u^{(1)}(s,m,h) \leftarrow 0$ for all $(s,m,h) \in \mathcal{J}(X)$
for $k = 1$ to K **do**
  $z^{(k)} \leftarrow \arg\max_{z \in \mathcal{Z}} (g(z) + \sum_{(s,m,h) \in \mathcal{J}(X)} u^{(k)}(s,m,h) z((s,m),h))$
  $Y^{(k)} \leftarrow \arg\max_{Y \in \mathcal{Y}(X)} (F(Y) - \sum_{(s,m,h) \in \mathcal{J}(X)} u^{(k)}(s,m,h) Y_s(m,h))$
  if $Y_s^{(k)}(m,h) = z^{(k)}((s,m),h)$
    for all $(s,m,h) \in \mathcal{J}(X)$ **then**
    return $(Y^{(k)}, z^{(k)})$
  for all $(s,m,h) \in \mathcal{J}(X)$,
    $u^{(k+1)}(s,m,h) \leftarrow u^{(k)}(s,m,h) + \alpha_k(z^{(k)}((s,m),h) - Y_s^{(k)}(m,h))$
return $(Y^{(K)}, z^{(K)})$

Figure 3-3: The global decoding algorithm for dependency parsing models.

3.7 Experiments and Results

3.7.1 Experiments

We perform experiments on two important NLP tasks: dependency parsing and POS tagging. For both tasks, we experiment with state-of-the-art sentence-level models trained with small amounts of annotated data.

Our data was taken from the WSJ PennTreebank: training sentences from section 0, development sentences from section 22, and test sentences from section 23. For both parsing and tagging we train the sentence-level model with 50 sentences and then use it as part of the constrained test-time inference objective.

For dependency parsing we use the second-order projective MST parser [McDonald et al., 2005b] [1] and the gold-standard POS tags of the corpus. For POS tagging we use the Stanford POS tagger [Toutanova et al., 2003] [2].

In our dual decomposition inference algorithm, we use maximum iterations $K = 1000$ and tune the decay rate following the protocol described by Koo et al. [2010]. For convenience of experiments we divide the parsing test set into subsets of 100 sentences. For tagging, since we import a large number of additional sentences (see below), the actual test

[1] http://sourceforge.net/projects/mstparser/
[2] http://nlp.stanford.edu/software/tagger.shtml

set size is on the order of a hundred thousand sentences. We therefore divide it into sets of 1000 sentences each [3].

We now turn to the specific details of the inter-sentence constraints we use in our experiments.

Parsing Constraints. As mentioned in section 3.3, we impose parsing constraints among words that appear in the same context. An important question is how to select the neighborhood that makes up a context.

First define a context template to be a set of offsets $\{r, \ldots, s\}$ with $r \leq 0 \leq s$, which specifies the neighboring words to include in a context. In the example of figure 3-1, the context template $\{-1, 0, 1, 2\}$ applied to the word `girls/NNS` would produce the context `JJ NNS VBD RB`. For each word in the corpus, we consider all possible templates with $s - r < 4$. To select the best context for this word, we collect statistics from the training data and choose the context that predicts the POS of its head word with the highest probability.

Once we select the context of each word, we add a consensus node for each context type in the corpus. We connect each word node to its corresponding consensus node. We also add fixed nodes for every word in the training data, and connect these nodes to their corresponding consensus nodes. The final MRF has the topology shown in figure 3-2(d).

To score the labels in this MRF, we use the function $g = \theta_{spotts}$ defined in section 3.5. This scoring function rewards words in the same context for choosing head words with the same POS tag, and it prefers tags that are similar to tags that are entirely different. We consider two tags to be similar if they agree on the first two letters of their name.

POS Tagging Constraints. For POS tagging, our constraints focus on unknown words - words that are not observed at all in the training data. Since we only use a small amount of training data, these words make up a large portion of the total words in the test corpus.

To enforce consistency among word instances of the same type, we add a consensus node for each unknown type in the test corpus. Analogously to parsing, this node encourages consistency in the POS tags of the connected nodes. We use the same scoring function θ_{spotts} in this model.

[3] We note that an iteration of the algorithm requires only a single pass through the test data, one sentence at a time.Therefore, splitting the test data does not change the efficiency or memory usage of an iteration of the algorithm.

	Accuracy
MST Parser	69.5%
Inter-sentence Dependency Parser	72.0%

Table 3.1: Attachment accuracy results for dependency parsing on WSJ Section 23 with 50 sentences training data. MST Parser refers to the second-order, projective dependency parser of McDonald et al. [2005b]. Inter-sentence dependency parser augments this baseline with global constraints.

	Accuracy	Unk. Acc.
Stanford POS Tagger	80.4%	63.1%
Inter-sentence POS Tagger	82.6%	67.7%

Table 3.2: POS tag accuracy results on WSJ Section 23 with 50 sentences training data. Stanford POS tagger refers to the maximum entropy trigram tagger of Toutanova et al. [2003]. Our inter-sentence POS tagger augments this baseline with global constraints.

Unfortunately, in most test sets we are unlikely to see an unknown word more than once or twice, and so the consistency constraints are not always helpful. To fix this sparsity issue, we import additional unannotated sentences for each unknown word from the New York Times section of the NANC corpus [Graff, 1995]. These sentences give us additional information for unknown word types.

Additionally, we note that morphologically related words often have similar POS tags. We can exploit this relationship by connecting related word types to the same consensus node. We experimented with various morphological variants and found that connecting a word type with the type generated by appending the suffix "s" was most beneficial. For each unknown word type, we also import sentences for its morphologically related words.

3.7.2 Results

We compare the accuracy of dependency parsing with global constraints to the state-of-the-art sentence-level dependency parser of McDonald et al. [2005b]. We measure directed head attachment accuracy using the CONLL-X dependency parsing shared task evaluation script [Buchholz and Marsi, 2006]. The results are given in table 3.1. We show a 2.5% gain over the baseline for this task.

For POS tagging, we compare our inter-sentence model to the Stanford POS tagger, a state-of-the-art sentence-level tagger. We measure token-level POS accuracy for all the words in the corpus and also for unknown words (words not observed in the training data). The results are given in table 3.2. We show a 2.2% gain over the baseline for this task.

The other important properties of the algorithm are its exactness and efficiency. Since a dual decomposition algorithm is guaranteed to provide an exact solution when it converges, we can compute the fraction of test subsections that reached convergence to measure exactness. For dependency parsing, the algorithm finds an exact solution for 95.7% of the subsections. For POS tagging, the algorithm finds the exact solution for 99.8% of the subsections.

We measure the efficiency of the algorithm by the mean and median number of iterations. At worst, each iteration requires a single pass of the sentence-level model over the test data, plus a small cost for solving the MRF. For POS tagging, the mean number of iterations is 53.9 and the median is 53. For dependency parsing, the mean number of iterations is 136 and the median is 96.

3.8 Conclusion

In this chapter, we proposed a corpus-level objective that augments sentence-level models with inter-sentence consistency constraints. We describe an exact and efficient dual decomposition for decoding this objective. When applying our model to dependency parsing and part-of-speech tagging, we demonstrate a significant improvement over state-of-the-art sentence-level models. In future work, we intend explore efficient techniques for joint parameter learning for both the global MRF and the local models.

Chapter 4

Higher-Order Non-Projective Dependency Parsing

This chapter is joint work with Terry Koo, Michael Collins, Tommi Jaakkola, and David Sontag. An earlier version of the content in this chapter was published at EMNLP 2010 [Koo et al., 2010].

In the next two chapters, we transition from integrated algorithms incorporating individual solvers for subproblems to relaxation algorithms for specific decoding tasks. In this chapter, we describe a dual decomposition algorithm for higher-order non-projective dependency parsing. In the next chapter, we give a Lagrangian relaxation for syntactic machine translation.

4.1 Introduction

Non-projective dependency parsing is useful for many languages that exhibit non-projective syntactic structures. Unfortunately, the non-projective parsing problem is known to be NP-hard for all but the simplest models [McDonald and Satta, 2007]. This chapter introduces algorithms for non-projective parsing based on dual decomposition. We focus on parsing algorithms for *non-projective head automata*, a generalization of the head-automata models of Eisner [2000a] and Alshawi [1996] to non-projective structures. These models include non-projective dependency parsing models with higher-order (e.g., sibling and/or grandparent) dependency relations as a special case. Although decoding of full parse structures with non-

projective head automata is intractable, we leverage the observation that key components of the decoding *can* be efficiently computed using combinatorial algorithms. In particular,

1. Decoding for individual head-words can be accomplished using dynamic programming.
2. Decoding for arc-factored models can be accomplished using directed minimum-weight spanning tree (MST) algorithms.

In this chapter, we first give the definition for non-projective head automata, and describe the parsing algorithm. The algorithm can is an instance of Lagrangian relaxation; we describe this connection, and give convergence guarantees for the method. We describe a generalization to models that include grandparent dependencies. We then introduce a perceptron-driven training algorithm that makes use of point 1 above.

We describe experiments on non-projective parsing for a number of languages, and in particular compare the dual decomposition algorithm to approaches based on general-purpose linear programming (LP) or integer linear programming (ILP) solvers [Martins et al., 2009a]. The accuracy of our models is higher than previous work on a broad range of datasets. The method gives exact solutions to the decoding problem, together with a certificate of optimality, on over 98% of test examples for many of the test languages, with parsing times ranging between 0.021 seconds/sentence for the most simple languages/models, to 0.295 seconds/sentence for the most complex settings. The method compares favorably to previous work using LP/ILP formulations, both in terms of efficiency, and also in terms of the percentage of exact solutions returned.

While the focus of the current chapter is on non-projective dependency parsing, the approach opens up new ways of thinking about parsing algorithms for lexicalized formalisms such as TAG [Joshi and Schabes, 1997], CCG [Steedman, 2000], and projective head automata.

4.2 Sibling Models

This section describes a particular class of models, *sibling models*; the next section describes a dual-decomposition algorithm for decoding these models.

Consider the dependency parsing problem for a sentence with n words. We define the *index set* for dependency parsing to be $\mathcal{I} = \{(i, j) : i \in \{0 \ldots n\}, j \in \{1 \ldots n\}, i \neq j\}$. A

dependency parse is a vector $y = \{y(i,j) : (i,j) \in \mathcal{I}\}$, where $y(i,j) = 1$ if a dependency with head word i and modifier j is in the parse, 0 otherwise. We use $i = 0$ for the root symbol. We define $\mathcal{Y}$ to be the set of all well-formed non-projective dependency parses (i.e., the set of directed spanning trees rooted at node 0). Given a function $f : \mathcal{Y} \mapsto \mathcal{R}$ that assigns scores to parse trees, the optimal parse is

$$y^* = \arg\max_{y \in \mathcal{Y}} f(y) \tag{4.1}$$

A particularly simple definition of $f(y)$ is $f(y) = \sum_{(i,j) \in \mathcal{I}} y(i,j)\theta(i,j)$ where $\theta(i,j)$ is the score for dependency (i,j). Models with this form are often referred to as *arc-factored models*. In this case the optimal parse tree y^* can be found efficiently using MST algorithms [McDonald et al., 2005a].

This chapter describes algorithms that compute y^* for more complex definitions of $f(y)$; in this section, we focus on algorithms for models that capture interactions between sibling dependencies. To this end, we will find it convenient to define the following notation. Given a vector y, define

$$y_{|i} = \{y(i,j) : j = 1 \ldots n, j \neq i\}$$

Hence $y_{|i}$ specifies the set of modifiers to word i; note that the vectors $y_{|i}$ for $i = 0 \ldots n$ form a partition of the full set of variables.

We then assume that $f(y)$ takes the form

$$f(y) = \sum_{i=0}^{n} f_i(y_{|i}) \tag{4.2}$$

Thus $f(y)$ decomposes into a sum of terms, where each f_i considers modifiers to the i'th word alone.

In the general case, finding $y^* = \arg\max_{y \in \mathcal{Y}} f(y)$ under this definition of $f(y)$ is an NP-hard problem. However for certain definitions of f_i, it is possible to efficiently compute $\arg\max_{y_{|i} \in \mathcal{Z}_i} f_i(y_{|i})$ for any value of i, typically using dynamic programming. (Here we use $\mathcal{Z}_i$ to refer to the set of all possible values for $y_{|i}$: specifically, $\mathcal{Z}_0 = \{0,1\}^n$ and for $i \neq 0$, $\mathcal{Z}_i = \{0,1\}^{n-1}$.) In these cases we can efficiently compute

$$z^* = \arg\max_{z \in \mathcal{Z}} f(z) = \arg\max_{z \in \mathcal{Z}} \sum_i f_i(z_{|i}) \tag{4.3}$$

where $\mathcal{Z} = \{z : z_{|i} \in \mathcal{Z}_i \quad \text{for } i = 0 \ldots n\}$ by simply computing $z^*_{|i} = \arg\max_{z_{|i} \in \mathcal{Z}_i} f_i(z_{|i})$ for

$i = 0 \ldots n$. Eq. 4.3 can be considered to be an approximation to Eq. 4.1, where we have replaced $\mathcal{Y}$ with $\mathcal{Z}$. We will make direct use of this approximation in the dual decomposition parsing algorithm. Note that $\mathcal{Y} \subseteq \mathcal{Z}$, and in all but trivial cases, $\mathcal{Y}$ is a strict subset of $\mathcal{Z}$. For example, a structure $z \in \mathcal{Z}$ could have $z(i,j) = z(j,i) = 1$ for some (i,j); it could contain longer cycles; or it could contain words that do not modify exactly one head. Nevertheless, with suitably powerful functions f_i—for example functions based on discriminative models—z^* may be a good approximation to y^*. Later we will see that dual decomposition can effectively use MST inference to rule out ill-formed structures.

We now give the main assumption underlying sibling models:

Assumption 1 (Sibling Decompositions) *A model $f(y)$ satisfies the sibling-decomposition assumption if: 1) $f(y) = \sum_{i=0}^{n} f_i(y_{|i})$ for some set of functions $f_0 \ldots f_n$. 2) For any $i \in \{0 \ldots n\}$, for any value of the variables $u(i,j) \in \mathcal{R}$ for $j = 1 \ldots n$, it is possible to compute*

$$\arg \max_{y_{|i} \in \mathcal{Z}_i} \left(f_i(y_{|i}) - \sum_j u(i,j) y(i,j) \right)$$

in polynomial time.

The second condition includes additional terms involving $u(i,j)$ variables that modify the scores of individual dependencies. These terms are benign for most definitions of f_i, in that they do not alter decoding complexity. They will be of direct use in the dual decomposition parsing algorithm.

Example 1: Bigram Sibling Models. Recall that $y_{|i}$ is a binary vector specifying which words are modifiers to the head-word i. Define $l_1 \ldots l_p$ to be the sequence of left modifiers to word i under $y_{|i}$, and $r_1 \ldots r_q$ to be the set of right modifiers (e.g., consider the case where $n = 5$, $i = 3$, and we have $y(3,1) = y(3,5) = 0$, and $y(3,2) = y(3,4) = 1$: in this case $p = 1$, $l_1 = 2$, and $q = 1$, $r_1 = 4$). In *bigram sibling models*, we have

$$f_i(y_{|i}) = \sum_{k=1}^{p+1} g_L(i, l_{k-1}, l_k) + \sum_{k=1}^{q+1} g_R(i, r_{k-1}, r_k)$$

where $l_0 = r_0 = \text{START}$ is the initial state, and $l_{p+1} = r_{q+1} = \text{END}$ is the end state. The functions g_L and g_R assign scores to bigram dependencies to the left and right of the

head. Under this model calculating $\arg\max_{y_{|i} \in \mathcal{Z}_i} \left(f_i(y_{|i}) - \sum_j u(i,j) y(i,j) \right)$ takes $O(n^2)$ time using dynamic programming, hence the model satisfies Assumption 1. □

Example 2: Head Automata Head-automata models constitute a second important model type that satisfy the sibling-decomposition assumption (bigram sibling models are a special case of head automata). These models make use of functions $g_R(i, s, s', r)$ where $s \in S, s' \in S$ are variables in a set of possible states S, and r is an index of a word in the sentence such that $i < r \leq n$. The function g_R returns a cost for taking word r as the next dependency, and transitioning from state s to s'. A similar function g_L is defined for left modifiers. We define

$$f_i(y_{|i}, s_0 \dots s_q, t_0 \dots t_p) = \\ \sum_{k=1}^{q} g_R(i, s_{k-1}, s_k, r_k) + \sum_{k=1}^{p} g_L(i, t_{k-1}, t_k, l_l)$$

to be the joint score for dependencies $y_{|i}$, and left and right state sequences $s_0 \dots s_q$ and $t_0 \dots t_p$. We specify that $s_0 = t_0 = \text{START}$ and $s_q = t_p = \text{END}$. In this case we define

$$f_i(y_{|i}) = \max_{s_0 \dots s_q, t_0 \dots t_p} f_i(y_{|i}, s_0 \dots s_q, t_0 \dots t_p)$$

and it follows that $\arg\max_{y_{|i} \in \mathcal{Z}_i} f_i(y_{|i})$ can be computed in $O(n|S|^2)$ time using a variant of the Viterbi algorithm, hence the model satisfies the sibling-decomposition assumption. □

4.3 The Parsing Algorithm

We now describe the dual decomposition parsing algorithm for models that satisfy Assumption 1. Consider the following generalization of the decoding problem from Eq. 4.1, where $f(y) = \sum_i f_i(y_{|i})$, $h(y) = \sum_{(i,j) \in \mathcal{I}} \gamma(i,j) y(i,j)$, and $\gamma(i,j) \in \mathcal{R}$ for all (i,j):[1]

$$\arg \max_{z \in \mathcal{Z}, y \in \mathcal{Y}} \quad f(z) + h(y) \tag{4.4}$$

$$\text{such that} \quad z(i,j) = y(i,j) \ \text{ for all } (i,j) \in \mathcal{I} \tag{4.5}$$

[1]This is equivalent to Eq. 4.1 when $\gamma(i,j) = 0$ for all (i,j). In some cases, however, it is convenient to have a model with non-zero values for the γ variables; see the Appendix. Note that this definition of $h(y)$ allows $\arg\max_{y \in \mathcal{Y}} h(y)$ to be calculated efficiently, using MST inference.

Set $u^{(1)}(i,j) \leftarrow 0$ for all $(i,j) \in \mathcal{I}$
for $k = 1$ to K **do**

$$y^{(k)} \leftarrow \arg\max_{y \in \mathcal{Y}} \sum_{(i,j) \in \mathcal{I}} \left(\gamma(i,j) + u^{(k)}(i,j)\right) y(i,j)$$

for $i \in \{0 \ldots n\}$,

$$z_{|i}^{(k)} \leftarrow \arg\max_{z_{|i} \in \mathcal{Z}_i} \left(f_i(z_{|i}) - \sum_j u^{(k)}(i,j) z(i,j)\right)$$

if $y^{(k)}(i,j) = z^{(k)}(i,j)$ for all $(i,j) \in \mathcal{I}$ **then**
return $(y^{(k)}, z^{(k)})$
for all $(i,j) \in \mathcal{I}$,
$u^{(k+1)}(i,j) \leftarrow u^{(k)}(i,j) + \alpha_k (z^{(k)}(i,j) - y^{(k)}(i,j))$
return $(y^{(K)}, z^{(K)})$

Figure 4-1: The parsing algorithm for sibling decomposable models. $\alpha_k \geq 0$ for $k = 1 \ldots K$ are step sizes, see Appendix 6.2 for details.

Although the maximization w.r.t. z is taken over the set $\mathcal{Z}$, the constraints in Eq. 4.5 ensure that $z = y$ for some $y \in \mathcal{Y}$, and hence that $z \in \mathcal{Y}$.

Without the $z(i,j) = y(i,j)$ constraints, the objective would decompose into the separate maximizations $z^* = \arg\max_{z \in \mathcal{Z}} f(z)$, and $y^* = \arg\max_{y \in \mathcal{Y}} h(y)$, which can be easily solved using dynamic programming and MST, respectively. Thus, it is these constraints that complicate the optimization. Our approach gets around this difficulty by introducing new variables, $u(i,j)$, that serve to enforce agreement between the $y(i,j)$ and $z(i,j)$ variables. In the next section we will show that these $u(i,j)$ variables are actually Lagrange multipliers for the $z(i,j) = y(i,j)$ constraints.

Our parsing algorithm is shown in Figure 4-1. At each iteration k, the algorithm finds $y^{(k)} \in \mathcal{Y}$ using an MST algorithm, and $z^{(k)} \in \mathcal{Z}$ through separate decoding of the $(n+1)$ sibling models. The $u^{(k)}$ variables are updated if $y^{(k)}(i,j) \neq z^{(k)}(i,j)$ for some (i,j); these updates modify the objective functions for the two decoding steps, and intuitively encourage the $y^{(k)}$ and $z^{(k)}$ variables to be equal.

4.3.1 Lagrangian Relaxation

Recall that the main difficulty in solving Eq. 4.4 was the $z = y$ constraints. We deal with these constraints using Lagrangian relaxation. We first introduce Lagrange multipliers

$u = \{u(i,j) : (i,j) \in \mathcal{I}\}$, and define the Lagrangian

$$\begin{aligned} L(u,y,z) \quad &= \\ & f(z) + h(y) + \sum_{(i,j)\in\mathcal{I}} u(i,j)\Big(y(i,j) - z(i,j)\Big) \end{aligned} \tag{4.6}$$

If L^* is the optimal value of Eq. 4.4 subject to the constraints in Eq. 4.5, then for any value of u,

$$L^* = \max_{z\in\mathcal{Z}, y\in\mathcal{Y}, y=z} L(u,y,z) \tag{4.7}$$

This follows because if $y = z$, the right term in Eq. 4.6 is zero for any value of u. The dual objective $L(u)$ is obtained by omitting the $y = z$ constraint:

$$\begin{aligned} L(u) \quad &= \quad \max_{z\in\mathcal{Z}, y\in\mathcal{Y}} L(u,y,z) \\ &= \quad \max_{z\in\mathcal{Z}}\Big(f(z) - \sum_{i,j} u(i,j)z(i,j)\Big) \\ &\quad + \max_{y\in\mathcal{Y}}\Big(h(y) + \sum_{i,j} u(i,j)y(i,j)\Big). \end{aligned}$$

Since $L(u)$ maximizes over a larger space (y may not equal z), we have that $L^* \leq L(u)$ (compare this to Eq. 4.7). The *dual problem*, which our algorithm optimizes, is to obtain the tightest such upper bound,

$$\text{(Dual problem)} \qquad \min_{u\in\mathcal{R}^{|\mathcal{I}|}} L(u). \tag{4.8}$$

The dual objective $L(u)$ is convex, but not differentiable. However, we can use a subgradient method to derive an algorithm that is similar to gradient descent, and which minimizes $L(u)$. A subgradient of a convex function $L(u)$ at u is a vector d_u such that for all $v \in \mathcal{R}^{|\mathcal{I}|}$, $L(v) \geq L(u) + d_u \cdot (v - u)$. By standard results,

$$d_{u^{(k)}} = y^{(k)} - z^{(k)}$$

is a subgradient for $L(u)$ at $u = u^{(k)}$, where $z^{(k)} = \arg\max_{z\in\mathcal{Z}} f(z) - \sum_{i,j} u^{(k)}(i,j)z(i,j)$ and $y^{(k)} = \arg\max_{y\in\mathcal{Y}} h(y) + \sum_{i,j} u^{(k)}(i,j)y(i,j)$. Subgradient optimization methods are

iterative algorithms with updates that are similar to gradient descent:

$$u^{(k+1)} = u^{(k)} - \alpha_k d_{u^{(k)}} = u^{(k)} - \alpha_k (y^{(k)} - z^{(k)}),$$

where α_k is a step size. It is easily verified that the algorithm in Figure 4-1 uses precisely these updates.

4.3.2 Formal Guarantees

With an appropriate choice of the step sizes α_k, the subgradient method can be shown to solve the dual problem, i.e.

$$\lim_{k \to \infty} L(u^{(k)}) = \min_u L(u).$$

See Korte and Vygen [2008], page 120, for details.

As mentioned before, the dual provides an upper bound on the optimum of the primal problem (Eq. 4.4),

$$\max_{z \in \mathcal{Z}, y \in \mathcal{Y}, y=z} \quad f(z) + h(y) \leq \min_{u \in \mathcal{R}^{|\mathcal{I}|}} L(u). \tag{4.9}$$

However, we do not necessarily have strong duality—i.e., equality in the above equation—because the sets $\mathcal{Z}$ and $\mathcal{Y}$ are discrete sets. That said, for some functions $h(y)$ and $f(z)$ strong duality does hold, as stated in the following:

Theorem 5 *If for some $k \in \{1 \dots K\}$ in the algorithm in Figure 4-1, $y^{(k)}(i,j) = z^{(k)}(i,j)$ for all $(i,j) \in \mathcal{I}$, then $(y^{(k)}, z^{(k)})$ is a solution to the maximization problem in Eq. 4.4.*

Proof. We have that $f(z^{(k)}) + h(y^{(k)}) = L(u^{(k)}, z^{(k)}, y^{(k)}) = L(u^{(k)})$, where the last equality is because $y^{(k)}$, $z^{(k)}$ are defined as the respective arg max's. Thus, the inequality in Eq. 4.9 is tight, and $(y^{(k)}, z^{(k)})$ and $u^{(k)}$ are primal and dual optimal. □

Although the algorithm is not guaranteed to satisfy $y^{(k)} = z^{(k)}$ for some k, by Theorem 5 if it does reach such a state, then we have the guarantee of an *exact* solution to Eq. 4.4, with the dual solution u providing a certificate of optimality. We show in the experiments that this occurs very frequently, in spite of the parsing problem being NP-hard.

It can be shown that Eq. 4.8 is the dual of an LP relaxation of the original problem. When the conditions of Theorem 5 are satisfied, it means that the LP relaxation is *tight* for this instance. For brevity we omit the details, except to note that when the LP relaxation is

not tight, the optimal primal solution to the LP relaxation could be recovered by averaging methods [Nedić and Ozdaglar, 2009].

4.4 Grandparent Dependency Models

In this section we extend the approach to consider grandparent relations. In grandparent models each parse tree y is represented as a vector

$$y = \{y(i,j) : (i,j) \in \mathcal{I}\} \cup \{y_\uparrow(i,j) : (i,j) \in \mathcal{I}\}$$

where we have added a second set of duplicate variables, $y_\uparrow(i,j)$ for all $(i,j) \in \mathcal{I}$. The set of all valid parse trees is then defined as

$$\begin{aligned}\mathcal{Y} = \{y \quad : \quad & y(i,j) \text{ variables form a directed tree,} \\ & y_\uparrow(i,j) = y(i,j) \text{ for all } (i,j) \in \mathcal{I}\}\end{aligned}$$

We again partition the variables into $n+1$ subsets, $y_{|0} \ldots y_{|n}$, by (re)defining

$$\begin{aligned}y_{|i} \quad = \quad & \{y(i,j) : j = 1 \ldots n, j \neq i\} \\ & \cup \{y_\uparrow(k,i) : k = 0 \ldots n, k \neq i\}\end{aligned}$$

So as before $y_{|i}$ contains variables $y(i,j)$ which indicate which words modify the i'th word. In addition, $y_{|i}$ includes $y_\uparrow(k,i)$ variables that indicate the word that word i itself modifies.

The set of all possible values of $y_{|i}$ is now

$$\begin{aligned}\mathcal{Z}_i = \{y_{|i} : & y(i,j) \in \{0,1\} \text{ for } j = 1 \ldots n, j \neq i; \\ & y_\uparrow(k,i) \in \{0,1\} \text{ for } k = 0 \ldots n, k \neq i; \\ & \sum_k y_\uparrow(k,i) = 1\}\end{aligned}$$

Hence the $y(i,j)$ variables can take any values, but only one of the $y_\uparrow(k,i)$ variables can be equal to 1 (as only one word can be a parent of word i). As before, we define $\mathcal{Z} = \{y : y_{|i} \in \mathcal{Z}_i \text{ for } i = 0 \ldots n\}$.

We introduce the following assumption:

Assumption 2 (GS Decompositions)

A model $f(y)$ satisfies the grandparent/sibling-decomposition (GSD) assumption if: 1) $f(z) = \sum_{i=0}^{n} f_i(z_{|i})$ for some set of functions $f_0 \ldots f_n$. 2) For any $i \in \{0 \ldots n\}$, for any value of the variables $u(i,j) \in \mathcal{R}$ for $j = 1 \ldots n$, and $v(k,i) \in \mathcal{R}$ for $k = 0 \ldots n$, it is possible to compute

$$\arg \max_{z_{|i} \in \mathcal{Z}_i} (f_i(z_{|i}) - \sum_j u(i,j) z(i,j) - \sum_k v(k,i) z_{\uparrow}(k,i))$$

in polynomial time.

Again, it follows that we can approximate $y^* = \arg\max_{y \in \mathcal{Y}} \sum_{i=0}^{n} f_i(y_{|i})$ by $z^* = \arg\max_{z \in \mathcal{Z}} \sum_{i=0}^{n} f_i(z_{|i})$, by defining $z^*_{|i} = \arg\max_{z_{|i} \in \mathcal{Z}_i} f_i(z_{|i})$ for $i = 0 \ldots n$. The resulting vector z^* may be deficient in two respects. First, the variables $z^*(i,j)$ may not form a well-formed directed spanning tree. Second, we may have $z^*_{\uparrow}(i,j) \neq z^*(i,j)$ for some values of (i,j).

Example 3: Grandparent/Sibling Models An important class of models that satisfy Assumption 2 are defined as follows. Again, for a vector $y_{|i}$ define $l_1 \ldots l_p$ to be the sequence of left modifiers to word i under $y_{|i}$, and $r_1 \ldots r_q$ to be the set of right modifiers. Define k^* to the value for k such that $y_{\uparrow}(k,i) = 1$. Then the model is defined as follows:

$$f_i(y_{|i}) = \sum_{j=1}^{p+1} g_L(i, k^*, l_{j-1}, l_j) + \sum_{j=1}^{q+1} g_R(i, k^*, r_{j-1}, r_j)$$

This is very similar to the bigram-sibling model, but with the modification that the g_L and g_R functions depend in addition on the value for k^*. This allows these functions to model grandparent dependencies such as (k^*, i, l_j) and sibling dependencies such as (i, l_{j-1}, l_j). Finding $z^*_{|i}$ under the definition can be accomplished in $O(n^3)$ time, by decoding the model using dynamic programming separately for each of the $O(n)$ possible values of k^*, and picking the value for k^* that gives the maximum value under these decodings. □

A dual-decomposition algorithm for models that satisfy the GSD assumption is shown in Figure 4-2. The algorithm can be justified as an instance of Lagrangian relaxation applied to the problem

$$\arg \max_{z \in \mathcal{Z}, y \in \mathcal{Y}} \quad f(z) + h(y) \tag{4.10}$$

with constraints

$$z(i,j) = y(i,j) \text{ for all } (i,j) \in \mathcal{I} \tag{4.11}$$

Set $u^{(1)}(i,j) \leftarrow 0$, $v^{(1)}(i,j) \leftarrow 0$ for all $(i,j) \in \mathcal{I}$
for $k = 1$ to K **do**

$$y^{(k)} \leftarrow \arg\max_{y \in \mathcal{Y}} \sum_{(i,j) \in \mathcal{I}} y(i,j)\theta(i,j)$$

where $\theta(i,j) = \gamma(i,j) + u^{(k)}(i,j) + v^{(k)}(i,j)$.

for $i \in \{0 \ldots n\}$,

$$z_{|i}^{(k)} \leftarrow \arg\max_{z_{|i} \in \mathcal{Z}_i} \left(f_i(z_{|i}) - \sum_j u^{(k)}(i,j) z(i,j) - \sum_j v^{(k)}(j,i) z_{\uparrow}(j,i) \right)$$

if $y^{(k)}(i,j) = z^{(k)}(i,j) = z_{\uparrow}^{(k)}(i,j)$ for all $(i,j) \in \mathcal{I}$ **then**
return $(y^{(k)}, z^{(k)})$
for all $(i,j) \in \mathcal{I}$,
$u^{(k+1)}(i,j) \leftarrow u^{(k)}(i,j) + \alpha_k(z^{(k)}(i,j) - y^{(k)}(i,j))$
$v^{(k+1)}(i,j) \leftarrow v^{(k)}(i,j) + \alpha_k(z_{\uparrow}^{(k)}(i,j) - y^{(k)}(i,j))$
return $(y^{(K)}, z^{(K)})$

Figure 4-2: The parsing algorithm for grandparent/sibling-decomposable models.

$$z_{\uparrow}(i,j) = y(i,j) \text{ for all } (i,j) \in \mathcal{I} \tag{4.12}$$

The algorithm employs two sets of Lagrange multipliers, $u(i,j)$ and $v(i,j)$, corresponding to constraints in Eqs. 4.11 and 4.12. As in Theorem 5, if at any point in the algorithm $z^{(k)} = y^{(k)}$, then $(z^{(k)}, y^{(k)})$ is an exact solution to the problem in Eq. 4.10.

4.5 The Training Algorithm

In our experiments we make use of discriminative linear models, where for an input sentence x, the score for a parse y is $f(y) = w \cdot \phi(x,y)$ where $w \in \mathcal{R}^d$ is a parameter vector, and $\phi(x,y) \in \mathcal{R}^d$ is a feature-vector representing parse tree y in conjunction with sentence x. We will assume that the features decompose in the same way as the sibling-decomposable or grandparent/sibling-decomposable models, that is $\phi(x,y) = \sum_{i=0}^{n} \phi(x, y_{|i})$ for some feature vector definition $\phi(x, y_{|i})$. In the *bigram sibling* models in our experiments, we assume that

$$\phi(x, y_{|i}) = \sum_{k=1}^{p+1} \phi_L(x, i, l_{k-1}, l_k) + \sum_{k=1}^{q+1} \phi_R(x, i, r_{k-1}, r_k)$$

where as before $l_1 \ldots l_p$ and $r_1 \ldots r_q$ are left and right modifiers under $y_{|i}$, and where ϕ_L and ϕ_R are feature vector definitions. In the *grandparent models* in our experiments, we use

a similar definition with feature vectors $\phi_L(x, i, k^*, l_{k-1}, l_k)$ and $\phi_R(x, i, k^*, r_{k-1}, r_k)$, where k^* is the parent for word i under $y_{|i}$.

We train the model using the averaged perceptron for structured problems [Collins, 2002a]. Given the i'th example in the training set, $(x^{(i)}, y^{(i)})$, the perceptron updates are as follows:

- $z^* = \arg\max_{y \in \mathcal{Z}} w \cdot \phi(x^{(i)}, y)$
- If $z^* \neq y^{(i)}$, $w = w + \phi(x^{(i)}, y^{(i)}) - \phi(x^{(i)}, z^*)$

The first step involves inference over the set $\mathcal{Z}$, rather than $\mathcal{Y}$ as would be standard in the perceptron. Thus, decoding during training can be achieved by dynamic programming over head automata alone, which is very efficient.

Our training approach is closely related to *local training methods* [Punyakanok et al., 2005]. We have found this method to be effective, very likely because $\mathcal{Z}$ is a superset of $\mathcal{Y}$. Our training algorithm is also related to recent work on training using *outer bounds* (see, e.g., [Taskar et al., 2003, Finley and Joachims, 2008, Kulesza and Pereira, 2008, Martins et al., 2009a]). Note, however, that the LP relaxation optimized by dual decomposition is significantly tighter than $\mathcal{Z}$. Thus, an alternative approach would be to use the dual decomposition algorithm for inference during training.

4.6 Experiments

We report results on a number of data sets. For comparison to Martins et al. [2009a], we perform experiments for Danish, Dutch, Portuguese, Slovene, Swedish and Turkish data from the CoNLL-X shared task [Buchholz and Marsi, 2006], and English data from the CoNLL-2008 shared task [Surdeanu et al., 2008]. We use the official training/test splits for these data sets, and the same evaluation methodology as Martins et al. [2009a]. For comparison to Smith and Eisner [2008a], we also report results on Danish and Dutch using their alternate training/test split. Finally, we report results on the English WSJ treebank, and the Prague treebank. We use feature sets that are very similar to those described in Carreras [2007a]. We use marginal-based pruning, using marginals calculated from an arc-factored spanning tree model using the matrix-tree theorem [McDonald and Satta, 2007, Smith and Smith, 2007, Koo et al., 2007].

In all of our experiments we set the value K, the maximum number of iterations of dual decomposition in Figures 4-1 and 4-2, to be 5,000. If the algorithm does not terminate—i.e.,

	Ma09	MST	Sib	G+S	Best	CertS	CertG	TimeS	TimeG	TrainS	TrainG
Dan	91.18	89.74	91.08	**91.78**	91.54	99.07	98.45	0.053	0.169	0.051	0.109
Dut	85.57	82.33	84.81	**85.81**	85.57	98.19	97.93	0.035	0.120	0.046	0.048
Por	92.11	90.68	92.57	**93.03**	92.11	99.65	99.31	0.047	0.257	0.077	0.103
Slo	85.61	82.39	84.89	**86.21**	85.61	90.55	95.27	0.158	0.295	0.054	0.130
Swe	90.60	88.79	90.10	**91.36**	90.60	98.71	98.97	0.035	0.141	0.036	0.055
Tur	76.34	75.66	77.14	**77.55**	76.36	98.72	99.04	0.021	0.047	0.016	0.036
Eng[1]	91.16	89.20	91.18	**91.59**	—	98.65	99.18	0.082	0.200	0.032	0.076
Eng[2]	—	90.29	92.03	**92.57**	—	98.96	99.12	0.081	0.168	0.032	0.076
	Sm08	MST	Sib	G+S	—	CertS	CertG	TimeS	TimeG	TrainS	TrainG
Dan	86.5	87.89	89.58	**91.00**	—	98.50	98.50	0.043	0.120	0.053	0.065
Dut	88.5	88.86	90.87	**91.76**	—	98.00	99.50	0.036	0.046	0.050	0.054
	Mc06	MST	Sib	G+S	—	CertS	CertG	TimeS	TimeG	TrainS	TrainG
PTB	91.5	90.10	91.96	**92.46**	—	98.89	98.63	0.062	0.210	0.028	0.078
PDT	85.2	84.36	86.44	**87.32**	—	96.67	96.43	0.063	0.221	0.019	0.051

Table 4.1: A comparison of non-projective automaton-based parsers with results from previous work. MST: Our first-order baseline. Sib/G+S: Non-projective head automata with sibling or grandparent/sibling interactions, decoded via dual decomposition. Ma09: The best UAS of the LP/ILP-based parsers introduced in Martins et al. [2009a]. Sm08: The best UAS of any LBP-based parser in Smith and Eisner [2008a]. Mc06: The best UAS reported by McDonald and Pereira [2006]. Best: For the CoNLL-X languages only, the best UAS for any parser in the original shared task [Buchholz and Marsi, 2006] or in any column of Martins et al. [2009a]; note that the latter includes McDonald and Pereira [2006], Nivre and McDonald [2008], and Martins et al. [2008]. CertS/CertG: Percent of test examples for which dual decomposition produced a certificate of optimality, for Sib/G+S. TimeS/TimeG: Seconds/sentence for test decoding, for Sib/G+S. TrainS/TrainG: Seconds/sentence during training, for Sib/G+S. For consistency of timing, test decoding was carried out on identical machines with zero additional load; however, training was conducted on machines with varying hardware and load. We ran two tests on the CoNLL-08 corpus. Eng[1]: UAS when testing on the CoNLL-08 validation set, following Martins et al. [2009a]. Eng[2]: UAS when testing on the CoNLL-08 test set.

it does not return $(y^{(k)}, z^{(k)})$ within 5,000 iterations—we simply take the parse $y^{(k)}$ with the maximum value of $f(y^{(k)})$ as the output from the algorithm. At first sight 5,000 might appear to be a large number, but decoding is still fast—see Sections 4.6.3 and 4.6.4 for discussion.[2] The strategy for choosing step sizes α_k is described in Appendix 6.2, along with other details.

We first discuss performance in terms of *accuracy*, *success in recovering an exact solution*, and *parsing speed*. We then describe additional experiments examining various aspects of the algorithm.

[2]Note also that the feature vectors ϕ and inner products $w \cdot \phi$ only need to be computed once, thus saving computation.

Sib	Acc	Int	Time	Rand
LP(S)	92.14	88.29	0.14	11.7
LP(M)	92.17	93.18	0.58	30.6
ILP	92.19	100.0	1.44	100.0
DD-5000	92.19	98.82	0.08	35.6
DD-250	92.23	89.29	0.03	10.2
G+S	Acc	Int	Time	Rand
LP(S)	92.60	91.64	0.23	0.0
LP(M)	92.58	94.41	0.75	0.0
ILP	92.70	100.0	1.79	100.0
DD-5000	92.71	98.76	0.23	6.8
DD-250	92.66	85.47	0.12	0.0

Table 4.2: A comparison of dual decomposition with linear programs described by Martins et al. [2009a]. LP(S): Linear Program relaxation based on single-commodity flow. LP(M): Linear Program relaxation based on multi-commodity flow. ILP: Exact Integer Linear Program. DD-5000/DD-250: Dual decomposition with non-projective head automata, with $K = 5000/250$. Upper results are for the sibling model, lower results are G+S. Columns give scores for UAS accuracy, percentage of solutions which are integral, and solution speed in seconds per sentence. These results are for Section 22 of the PTB. The last column is the percentage of integral solutions on a random problem of length 10 words. The (I)LP experiments were carried out using Gurobi, a high-performance commercial-grade solver.

4.6.1 Accuracy

Table 4.1 shows results for previous work on the various data sets, and results for an arc-factored model with pure MST decoding with our features. (We use the acronym UAS (unlabeled attachment score) for dependency accuracy.) We also show results for the bigram-sibling and grandparent/sibling (G+S) models under dual decomposition. Both the bigram-sibling and G+S models show large improvements over the arc-factored approach; they also compare favorably to previous work—for example the G+S model gives better results than all results reported in the CoNLL-X shared task, on all languages. Note that we use different feature sets from both Martins et al. [2009a] and Smith and Eisner [2008a].

4.6.2 Success in Recovering Exact Solutions

Next, we consider how often our algorithms return an exact solution to the original optimization problem, with a certificate—i.e., how often the algorithms in Figures 4-1 and 4-2 terminate with $y^{(k)} = z^{(k)}$ for some value of $k < 5000$ (and are thus optimal, by Theorem 5). The CertS and CertG columns in Table 4.1 give the results for the sibling and G+S models

respectively. For all but one setting[3] over 95% of the test sentences are decoded exactly, with 99% exactness in many cases.

For comparison, we also ran both the single-commodity flow and multiple-commodity flow LP relaxations of Martins et al. [2009a] with our models and features. We measure how often these relaxations terminate with an exact solution. The results in Table 4.2 show that our method gives exact solutions more often than both of these relaxations.[4] In computing the accuracy figures for Martins et al. [2009a], we project fractional solutions to a well-formed spanning tree, as described in that paper.

Finally, to better compare the tightness of our LP relaxation to that of earlier work, we consider randomly-generated instances. Table 4.2 gives results for our model and the LP relaxations of Martins et al. [2009a] with randomly generated scores on automata transitions. We again recover exact solutions more often than the Martins et al. relaxations. Note that with random parameters the percentage of exact solutions is significantly lower, suggesting that the exactness of decoding of the trained models is a special case. We speculate that this is due to the high performance of approximate decoding with $\mathcal{Z}$ in place of $\mathcal{Y}$ under the trained models for f_i; the training algorithm described in section 4.5 may have the tendency to make the LP relaxation tight.

4.6.3 Speed

Table 4.1, columns TimeS and TimeG, shows decoding times for the dual decomposition algorithms. Table 4.2 gives speed comparisons to Martins et al. [2009a]. Our method gives significant speed-ups over the Martins et al. [2009a] method, presumably because it leverages the underlying structure of the problem, rather than using a generic solver.

4.6.4 Lazy Decoding

Here we describe an important optimization in the dual decomposition algorithms. Consider the algorithm in Figure 4-1. At each iteration we must find

$$z_{|i}^{(k)} = \arg\max_{z_{|i} \in \mathcal{Z}_i} (f_i(z_{|i}) - \sum_j u^{(k)}(i,j) z(i,j))$$

[3]The exception is Slovene, which has the smallest training set at only 1534 sentences.

[4]Note, however, that it is possible that the Martins et al. relaxations would have given a higher proportion of integral solutions if their relaxation was used during training.

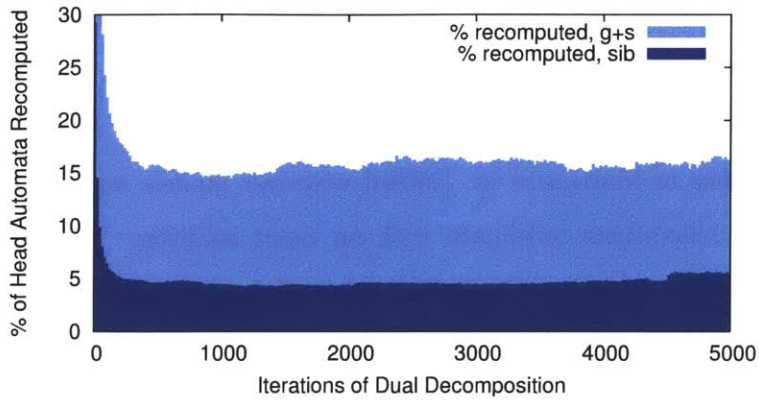


Figure 4-3: The average percentage of head automata that must be recomputed on each iteration of dual decomposition on the PTB validation set.

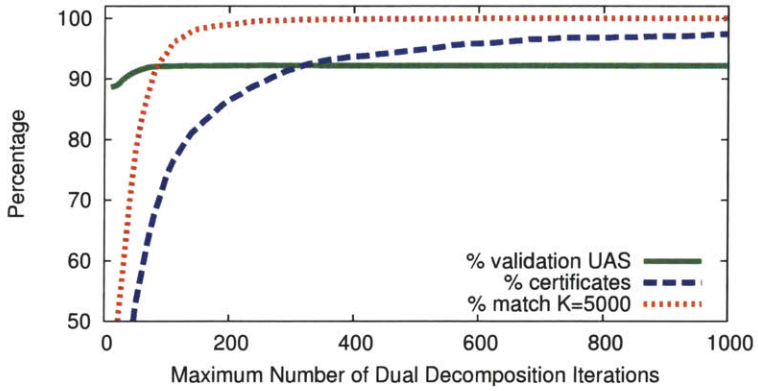


Figure 4-4: The behavior of the dual-decomposition parser with sibling automata as the value of K is varied.

for $i = 0 \ldots n$. However, if for some i, $u^{(k)}(i,j) = u^{(k-1)}(i,j)$ for all j, then $z_{|i}^{(k)} = z_{|i}^{(k-1)}$. In *lazy decoding* we immediately set $z_{|i}^{(k)} = z_{|i}^{(k-1)}$ if $u^{(k)}(i,j) = u^{(k-1)}(i,j)$ for all j; this check takes $O(n)$ time, and saves us from decoding with the i'th automaton. In practice, the updates to u are very sparse, and this condition occurs very often in practice. Figure 4-3 demonstrates the utility of this method for both sibling automata and G+S automata.

4.6.5 Early Stopping

We also ran experiments varying the value of K—the maximum number of iterations—in the dual decomposition algorithms. As before, if we do not find $y^{(k)} = z^{(k)}$ for some value of $k \leq K$, we choose the $y^{(k)}$ with optimal value for $f(y^{(k)})$ as the final solution. Figure 4-4 shows three graphs: 1) the accuracy of the parser on PTB validation data versus the value for K; 2) the percentage of examples where $y^{(k)} = z^{(k)}$ at some point during the algorithm,

	Sib	P-Sib	G+S	P-G+S
PTB	92.19	92.34	92.71	92.70
PDT	86.41	85.67	87.40	86.43

Table 4.3: UAS of projective and non-projective decoding for the English (PTB) and Czech (PDT) validation sets. Sib/G+S: as in Table 4.1. P-Sib/P-G+S: Projective versions of Sib/G+S, where the MST component has been replaced with the Eisner [2000a] first-order projective parser.

hence the algorithm returns a certificate of optimality; 3) the percentage of examples where the solution returned is the same as the solution for the algorithm with $K = 5000$ (our original setting). It can be seen for K as small as 250 we get very similar accuracy to $K = 5000$ (see Table 4.2). In fact, for this setting the algorithm returns the same solution as for $K = 5000$ on 99.59% of the examples. However only 89.29% of these solutions are produced with a certificate of optimality ($y^{(k)} = z^{(k)}$).

4.6.6 How Good is the Approximation z^*?

We ran experiments measuring the quality of $z^* = \arg\max_{z \in \mathcal{Z}} f(z)$, where $f(z)$ is given by the perceptron-trained bigram-sibling model. Because z^* may not be a well-formed tree with n dependencies, we report precision and recall rather than conventional dependency accuracy. Results on the PTB validation set were 91.11%/88.95% precision/recall, which is accurate considering the unconstrained nature of the predictions. Thus the z^* approximation is clearly a good one; we suspect that this is one reason for the good convergence results for the method.

4.6.7 Importance of Non-Projective Decoding

It is simple to adapt the dual-decomposition algorithms in figures 4-1 and 4-2 to give *projective* dependency structures: the set $\mathcal{Y}$ is redefined to be the set of all projective structures, with the arg max over $\mathcal{Y}$ being calculated using a projective first-order parser [Eisner, 2000a]. Table 4.3 shows results for projective and non-projective parsing using the dual decomposition approach. For Czech data, where non-projective structures are common, non-projective decoding has clear benefits. In contrast, there is little difference in accuracy between projective and non-projective decoding on English.

4.7 Conclusions

We have described dual decomposition algorithms for non-projective parsing, which leverage existing dynamic programming and MST algorithms. There are a number of possible areas for future work. As described in section 4.6.7, the algorithms can be easily modified to consider projective structures by replacing $\mathcal{Y}$ with the set of projective trees, and then using first-order dependency parsing algorithms in place of MST decoding. This method could be used to derive parsing algorithms that include higher-order features, as an alternative to specialized dynamic programming algorithms. Eisner [2000a] describes extensions of head automata to include word senses; we have not discussed this issue in the current paper, but it is simple to develop dual decomposition algorithms for this case, using similar methods to those used for the grandparent models. The general approach should be applicable to other lexicalized syntactic formalisms, and potentially also to decoding in syntax-driven translation. In addition, our dual decomposition approach is well-suited to parallelization. For example, each of the head-automata could be optimized independently in a multi-core or GPU architecture. Finally, our approach could be used with other structured learning algorithms, e.g. Meshi et al. [2010].

Chapter 5

Syntactic Machine Translation

5.1 Introduction

In this chapter, we present an Lagrangian relaxation algorithm for decoding syntactic machine translation. Recent work has seen widespread use of synchronous probabilistic grammars in statistical machine translation (SMT). The decoding problem for a broad range of these systems (e.g., [Chiang, 2005, Marcu et al., 2006, Shen et al., 2008]) corresponds to the intersection of a (weighted) hypergraph with an n-gram language model.[1] The hypergraph represents a large set of possible translations, and is created by applying a synchronous grammar to the source language string. The language model is then used to rescore the translations in the hypergraph.

Decoding with these models is challenging, largely because of the cost of integrating an n-gram language model into the search process. Exact dynamic programming algorithms for the problem are well known [Bar-Hillel et al., 1964], but are too expensive to be used in practice.[2] Previous work on decoding for syntax-based SMT has therefore been focused primarily on approximate search methods.

This chapter describes an efficient algorithm for exact decoding of synchronous grammar models for translation. We avoid the construction of Bar-Hillel et al. [1964] by using *Lagrangian relaxation* to decompose the decoding problem into the following sub-problems:

1. Dynamic programming over the weighted hypergraph. This step does not require

[1] This problem is also relevant to other areas of statistical NLP, for example NL generation [Langkilde, 2000].

[2] E.g., with a trigram language model they run in $O(|E|w^6)$ time, where $|E|$ is the number of edges in the hypergraph, and w is the number of distinct lexical items in the hypergraph.

language model integration, and hence is highly efficient.

2. Application of an all-pairs shortest path algorithm to a directed graph derived from the weighted hypergraph. The size of the derived directed graph is linear in the size of the hypergraph, hence this step is again efficient.

Informally, the first decoding algorithm incorporates the weights and hard constraints on translations from the synchronous grammar, while the second decoding algorithm is used to integrate language model scores. Lagrange multipliers are used to enforce agreement between the structures produced by the two decoding algorithms.

In this chapter we first give background on hypergraphs and the decoding problem. We then describe our decoding algorithm. The algorithm uses a subgradient method to minimize a dual function. The dual corresponds to a particular linear programming (LP) relaxation of the original decoding problem. The method will recover an exact solution, with a certificate of optimality, if the underlying LP relaxation has an integral solution. In some cases, however, the underlying LP will have a fractional solution, in which case the method will not be exact. The second technical contribution of this chapter is to describe a method that iteratively tightens the underlying LP relaxation until an exact solution is produced. We do this by gradually introducing constraints to step 1 (dynamic programming over the hypergraph), while still maintaining efficiency.

We report experiments using the tree-to-string model of [Huang and Mi, 2010]. Our method gives exact solutions on over 97% of test examples. The method is comparable in speed to state-of-the-art decoding algorithms; for example, over 70% of the test examples are decoded in 2 seconds or less. We compare our method to cube pruning [Chiang, 2007], and find that our method gives improved model scores on a significant number of examples. One consequence of our work is that we give accurate estimates of the number of search errors for cube pruning.

5.2 Background: Hypergraphs

Translation with many syntax-based systems (e.g., [Chiang, 2005, Marcu et al., 2006, Shen et al., 2008, Huang and Mi, 2010]) can be implemented as a two-step process. The first step is to take an input sentence in the source language, and from this to create a hypergraph (sometimes called a translation forest) that represents the set of possible translations (strings

in the target language) and derivations under the grammar. The second step is to integrate an n-gram language model with this hypergraph. For example, in the system of [Chiang, 2005], the hypergraph is created as follows: first, the source side of the synchronous grammar is used to create a parse forest over the source language string. Second, transduction operations derived from synchronous rules in the grammar are used to create the target-language hypergraph. Chiang's method uses a synchronous context-free grammar, but the hypergraph formalism is applicable to a broad range of other grammatical formalisms, for example dependency grammars (e.g., [Shen et al., 2008]).

A hypergraph is a pair (V, E) where $V = \{1, 2, \ldots, |V|\}$ is a set of vertices, and E is a set of hyperedges. A single distinguished vertex is taken as the root of the hypergraph; without loss of generality we take this vertex to be $v = 1$. Each hyperedge $e \in E$ is a tuple $\langle \langle v_1, v_2, \ldots, v_k \rangle, v_0 \rangle$ where $v_0 \in V$, and $v_i \in \{2 \ldots |V|\}$ for $i = 1 \ldots k$. The vertex v_0 is referred to as the *head* of the edge. The ordered sequence $\langle v_1, v_2, \ldots, v_k \rangle$ is referred to as the *tail* of the edge; in addition, we sometimes refer to $v_1, v_2, \ldots v_k$ as the *children* in the edge. The number of children k may vary across different edges, but $k \geq 1$ for all edges (i.e., each edge has at least one child). We will use $h(e)$ to refer to the head of an edge e, and $t(e)$ to refer to the tail.

We will assume that the hypergraph is acyclic: intuitively this will mean that no derivation (as defined below) contains the same vertex more than once (see [Martin et al., 1990] for a formal definition).

Each vertex $v \in V$ is either a *non-terminal* in the hypergraph, or a *leaf*. The set of non-terminals is

$$V_N = \{v \in V : \exists e \in E \quad \text{such that} \quad h(e) = v\}$$

Conversely, the set of leaves is defined as

$$V_L = \{v \in V : \nexists e \in E \quad \text{such that} \quad h(e) = v\}$$

Finally, we assume that each $v \in V$ has a label $l(v)$. The labels for leaves will be *words*, and will be important in defining strings and language model scores for those strings. The labels for non-terminal nodes will not be important for results in this chapter.[3]

We now turn to derivations. Define an *index set* $\mathcal{I} = V \cup E$. A derivation is represented

[3]They might for example be non-terminal symbols from the grammar used to generate the hypergraph.

by a vector $y = \{y_r : r \in \mathcal{I}\}$ where $y_v = 1$ if vertex v is used in the derivation, $y_v = 0$ otherwise (similarly $y_e = 1$ if edge e is used in the derivation, $y_e = 0$ otherwise). Thus y is a vector in $\{0,1\}^{|\mathcal{I}|}$. A valid derivation satisfies the following constraints:

- $y_1 = 1$ (the root must be in the derivation).
- For all $v \in V_N$, $y_v = \sum_{e:h(e)=v} y_e$.
- For all $v \in 2 \ldots |V|$, $y_v = \sum_{e:v \in t(e)} y_e$.

We use $\mathcal{Y}$ to refer to the set of valid derivations. The set $\mathcal{Y}$ is a subset of $\{0,1\}^{|\mathcal{I}|}$ (not all members of $\{0,1\}^{|\mathcal{I}|}$ will correspond to valid derivations).

Each derivation y in the hypergraph will imply an ordered sequence of leaves $v_1 \ldots v_n$. We use $s(y)$ to refer to this sequence. The *sentence* associated with the derivation is then $l(v_1) \ldots l(v_n)$.

In a weighted hypergraph problem, we assume a parameter vector $\theta = \{\theta_r : r \in \mathcal{I}\}$. The score for any derivation is $f(y) = \theta \cdot y = \sum_{r \in \mathcal{I}} \theta_r y_r$. Simple bottom-up dynamic programming—essentially the CKY algorithm—can be used to find $y^* = \arg\max_{y \in \mathcal{Y}} f(y)$ under these definitions.

The focus of this chapter will be to solve problems involving the integration of a k'th order language model with a hypergraph. In these problems, the score for a derivation is modified to be

$$f(y) = \sum_{r \in \mathcal{I}} \theta_r y_r + \sum_{i=k}^{n} \theta(v_{i-k+1}, v_{i-k+2}, \ldots, v_i) \tag{5.1}$$

where $v_1 \ldots v_n = s(y)$. The $\theta(v_{i-k+1}, \ldots, v_i)$ parameters score n-grams of length k. These parameters are typically defined by a language model, for example with $k = 3$ we would have $\theta(v_{i-2}, v_{i-1}, v_i) = \log p(l(v_i)|l(v_{i-2}), l(v_{i-1}))$. The problem is then to find $y^* = \arg\max_{y \in \mathcal{Y}} f(y)$ under this definition.

Throughout this chapter we make the following assumption when using a bigram language model:

Assumption 3 *(*Bigram start/end assumption.*) For any derivation y, with leaves $s(y) = v_1, v_2, \ldots, v_n$, it is the case that: (1) $v_1 = 2$ and $v_n = 3$; (2) the leaves 2 and 3 cannot appear at any other position in the strings $s(y)$ for $y \in \mathcal{Y}$; (3) $l(2) =$ `<s>` where `<s>` is the start symbol in the language model; (4) $l(3) =$ `</s>` where `</s>` is the end symbol.*

This assumption allows us to incorporate language model terms that depend on the start and end symbols. It also allows a clean solution for boundary conditions (the start/end of strings).[4]

5.3 A Simple Lagrangian Relaxation Algorithm

We now give a Lagrangian relaxation algorithm for integration of a hypergraph with a bigram language model, in cases where the hypergraph satisfies the following simplifying assumption:

Assumption 4 *(*The strict ordering assumption.*) For any two leaves v and w, it is either the case that: 1) for all derivations y such that v and w are both in the sequence $l(y)$, v precedes w; or 2) for all derivations y such that v and w are both in $l(y)$, w precedes v.*

Thus under this assumption, the relative ordering of any two leaves is fixed. This assumption is overly restrictive:[5] the next section describes an algorithm that does not require this assumption. However deriving the simple algorithm will be useful in developing intuition, and will lead directly to the algorithm for the unrestricted case.

5.3.1 A Sketch of the Algorithm

At a high level, the algorithm is as follows. We introduce Lagrange multipliers $u(v)$ for all $v \in V_L$, with initial values set to zero. The algorithm then involves the following steps: (1) For each leaf v, find the previous leaf w that maximizes the score $\theta(w, v) - u(w)$ (call this leaf $\alpha^*(v)$, and define $\alpha_v = \theta(\alpha^*(v), v) - u(\alpha^*(v))$). (2) find the highest scoring derivation using dynamic programming over the original (non-intersected) hypergraph, with leaf nodes having weights $\theta_v + \alpha_v + u(v)$. (3) If the output derivation from step 2 has the same set of bigrams as those from step 1, then we have an exact solution to the problem. Otherwise, the Lagrange multipliers $u(v)$ are modified in a way that encourages agreement of the two steps, and we return to step 1.

[4]The assumption generalizes in the obvious way to k'th order language models: e.g., for trigram models we assume that $v_1 = 2$, $v_2 = 3$, $v_n = 4$, $l(2) = l(3) =$ `<s>`, $l(4) =$ `</s>`.

[5]It is easy to come up with examples that violate this assumption: for example a hypergraph with edges $\langle\langle 4, 5\rangle, 1\rangle$ and $\langle\langle 5, 4\rangle, 1\rangle$ violates the assumption. The hypergraphs found in translation frequently contain alternative orderings such as this.

Steps 1 and 2 can be performed efficiently; in particular, we avoid the classical dynamic programming intersection, instead relying on dynamic programming over the original, simple hypergraph.

5.3.2 A Formal Description

We now give a formal description of the algorithm. Define $\mathcal{B} \subseteq V_L \times V_L$ to be the set of all ordered pairs $\langle v, w \rangle$ such that there is at least one derivation y with v directly preceding w in $s(y)$. Extend the bit-vector y to include variables $y(v, w)$ for $\langle v, w \rangle \in \mathcal{B}$ where $y(v, w) = 1$ if leaf v is followed by w in $s(y)$, 0 otherwise. We redefine the index set to be $\mathcal{I} = V \cup E \cup \mathcal{B}$, and define $\mathcal{Y} \subseteq \{0, 1\}^{|\mathcal{I}|}$ to be the set of all possible derivations. Under assumptions 3 and 4 above, $\mathcal{Y} = \{y : y \text{ satisfies constraints } \mathbf{C0}, \mathbf{C1}, \mathbf{C2}\}$ where the constraint definitions are:

- **(C0)** The y_v and y_e variables form a derivation in the hypergraph, as defined in section 5.2.
- **(C1)** For all $v \in V_L$ such that $v \neq 2$, $y_v = \sum_{w:\langle w,v \rangle \in \mathcal{B}} y(w, v)$.
- **(C2)** For all $v \in V_L$ such that $v \neq 3$, $y_v = \sum_{w:\langle v,w \rangle \in \mathcal{B}} y(v, w)$.

C1 states that each leaf in a derivation has exactly one in-coming bigram, and that each leaf not in the derivation has 0 incoming bigrams; **C2** states that each leaf in a derivation has exactly one out-going bigram, and that each leaf not in the derivation has 0 outgoing bigrams.[6]

The score of a derivation is now $f(y) = \theta \cdot y$, i.e.,

$$f(y) = \sum_v \theta_v y_v + \sum_e \theta_e y_e + \sum_{\langle v,w \rangle \in \mathcal{B}} \theta(v, w) y(v, w)$$

where $\theta(v, w)$ are scores from the language model. Our goal is to compute $y^* = \arg\max_{y \in \mathcal{Y}} f(y)$.

Next, define $\mathcal{Y}'$ as

$$\mathcal{Y}' = \{y : y \text{ satisfies constraints } \mathbf{C0} \text{ and } \mathbf{C1}\}$$

In this definition we have dropped the **C2** constraints. To incorporate these constraints, we use Lagrangian relaxation, with one Lagrange multiplier $u(v)$ for each constraint in **C2**.

[6]Recall that according to the bigram start/end assumption the leaves 2/3 are reserved for the start/end of the sequence $s(y)$, and hence do not have an incoming/outgoing bigram.

Initialization: Set $u^0(v) = 0$ for all $v \in V_L$
Algorithm: For $t = 1 \ldots T$:

- $y^t = \arg\max_{y \in \mathcal{Y}'} L(u^{t-1}, y)$
- **If** y^t satisfies constraints **C2**, **return** y^t,
  Else $\forall v \in V_L$, $u^t(v) =$
  $u^{t-1}(v) - \delta^t \left(y^t(v) - \sum_{w:\langle v,w \rangle \in \mathcal{B}} y^t(v,w) \right)$

Figure 5-1: A simple Lagrangian relaxation algorithm. $\delta^t > 0$ is the step size at iteration t.

The Lagrangian is

$$\begin{aligned} L(u,y) &= f(y) + \sum_v u(v)(y(v) - \sum_{w:\langle v,w \rangle \in \mathcal{B}} y(v,w)) \\ &= \beta \cdot y \end{aligned}$$

where $\beta_v = \theta_v + u(v)$, $\beta_e = \theta_e$, and $\beta(v,w) = \theta(v,w) - u(v)$.

The dual problem is to find $\min_u L(u)$ where

$$L(u) = \max_{y \in \mathcal{Y}'} L(u,y)$$

Figure 5-1 shows a *subgradient* method for solving this problem. At each point the algorithm finds $y^t = \arg\max_{y \in \mathcal{Y}'} L(u^{t-1}, y)$, where u^{t-1} are the Lagrange multipliers from the previous iteration. If y^t satisfies the **C2** constraints in addition to **C0** and **C1**, then it is returned as the output from the algorithm. Otherwise, the multipliers $u(v)$ are updated. Intuitively, these updates encourage the values of y_v and $\sum_{w:\langle v,w \rangle \in \mathcal{B}} y(v,w)$ to be equal; formally, these updates correspond to subgradient steps.

The main computational step at each iteration is to compute $\arg\max_{y \in \mathcal{Y}'} L(u^{t-1}, y)$ This step is easily solved, as follows (we again use β_v, β_e and $\beta(v_1, v_2)$ to refer to the parameter values that incorporate Lagrange multipliers):

- For all $v \in V_L$, define $\alpha^*(v) = \arg\max_{w:\langle w,v \rangle \in \mathcal{B}} \beta(w,v)$ and $\alpha_v = \beta(\alpha^*(v), v)$. For all $v \in V_N$ define $\alpha_v = 0$.
- Using dynamic programming, find values for the y_v and y_e variables that form a valid derivation, and that maximize

$f'(y) = \sum_v (\beta_v + \alpha_v) y_v + \sum_e \beta_e y_e$.

- Set $y(v, w) = 1$ iff $y(w) = 1$ and $\alpha^*(w) = v$.

The critical point here is that through our definition of $\mathcal{Y}'$, which ignores the **C2** constraints, we are able to do efficient search as just described. In the first step we compute the highest scoring incoming bigram for each leaf v. In the second step we use conventional dynamic programming over the hypergraph to find an optimal derivation that incorporates weights from the first step. Finally, we fill in the $y(v, w)$ values. Each iteration of the algorithm runs in $O(|E| + |\mathcal{B}|)$ time.

There are close connections between Lagrangian relaxation and linear programming relaxations. The most important formal results are: 1) for any value of u, $L(u) \geq f(y^*)$ (hence the dual value provides an upper bound on the optimal primal value); 2) under an appropriate choice of the step sizes δ^t, the subgradient algorithm is guaranteed to converge to the minimum of $L(u)$ (i.e., we will minimize the upper bound, making it as tight as possible); 3) if at any point the algorithm in figure 5-1 finds a y^t that satisfies the **C2** constraints, then this is guaranteed to be the optimal primal solution.

Unfortunately, this algorithm may fail to produce a good solution for hypergraphs where the strict ordering constraint does not hold. In this case it is possible to find derivations y that satisfy constraints **C0**, **C1**, **C2**, but which are invalid. As one example, consider a derivation with $s(y) = 2, 4, 5, 3$ and $y(2, 3) = y(4, 5) = y(5, 4) = 1$. The constraints are all satisfied in this case, but the bigram variables are invalid (e.g., they contain a cycle).

5.4 The Full Algorithm

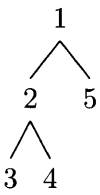
We now describe our full algorithm, which does not require the strict ordering constraint. In addition, the full algorithm allows a trigram language model. We first give a sketch, and then give a formal definition.

5.4.1 A Sketch of the Algorithm

A crucial idea in the new algorithm is that of *paths* between leaves in hypergraph derivations. Previously, for each derivation y, we had defined $s(y) = v_1, v_2, \ldots, v_n$ to be the sequence of leaves in y. In addition, we will define $g(y) = p_0, v_1, p_1, v_2, p_2, v_3, p_3, \ldots, p_{n-1}, v_n, p_n$ where

each p_i is a path in the derivation between leaves v_i and v_{i+1}. The path traces through the non-terminals that are between the two leaves in the tree.

As an example, consider the following derivation (with hyperedges $\langle\langle 2, 5\rangle, 1\rangle$ and $\langle\langle 3, 4\rangle, 2\rangle$):



For this example $g(y)$ is $\langle 1\downarrow, 2\downarrow\rangle$ $\langle 2\downarrow, 3\downarrow\rangle$ $\langle 3\downarrow\rangle$, 3, $\langle 3\uparrow\rangle$ $\langle 3\uparrow, 4\downarrow\rangle$ $\langle 4\downarrow\rangle$, 4, $\langle 4\uparrow\rangle$ $\langle 4\uparrow, 2\uparrow\rangle$ $\langle 2\uparrow, 5\downarrow\rangle$ $\langle 5\downarrow\rangle$, 5, $\langle 5\uparrow\rangle$ $\langle 5\uparrow, 1\uparrow\rangle$. States of the form $\langle a\downarrow\rangle$ and $\langle a\uparrow\rangle$ where a is a leaf appear in the paths respectively before/after the leaf a. States of the form $\langle a, b\rangle$ correspond to the steps taken in a top-down, left-to-right, traversal of the tree, where down and up arrows indicate whether a node is being visited for the first or second time (the traversal in this case would be $1, 2, 3, 4, 2, 5, 1$).

The mapping from a derivation y to a path $g(y)$ can be performed using the algorithm in figure 5-2. For a given derivation y, define $E(y) = \{y : y_e = 1\}$, and use $E(y)$ as the set of input edges to this algorithm. The output from the algorithm will be a set of states S, and a set of directed edges T, which together fully define the path $g(y)$.

In the simple algorithm, the first step was to predict the previous leaf for each leaf v, under a score that combined a language model score with a Lagrange multiplier score (i.e., compute $\arg\max_w \beta(w, v)$ where $\beta(w, v) = \theta(w, v) + u(w)$). In this section we describe an algorithm that for each leaf v again predicts the previous leaf, but in addition predicts the full *path* back to that leaf. For example, rather than making a prediction for leaf 5 that it should be preceded by leaf 4, we would also predict the path $\langle 4\uparrow\rangle\langle 4\uparrow, 2\uparrow\rangle$ $\langle 2\uparrow, 5\downarrow\rangle\langle 5\downarrow\rangle$ between these two leaves. Lagrange multipliers will be used to enforce consistency between these predictions (both paths and previous words) and a valid derivation.

5.4.2 A Formal Description

We first use the algorithm in figure 5-2 with the entire set of hyperedges, E, as its input. The result is a directed graph (S, T) that contains *all possible paths* for valid derivations in V, E (it also contains additional, ill-formed paths). We then introduce the following definition:

Input: A set E of hyperedges. **Output:** A directed graph S, T where S is a set of vertices, and T is a set of edges.
Step 1: Creating S: Define $S = \cup_{e \in E} S(e)$ where $S(e)$ is defined as follows. Assume $e = \langle \langle v_1, v_2, \ldots, v_k \rangle, v_0 \rangle$. Include the following states in $S(e)$: (1) $\langle v_0 \downarrow, v_1 \downarrow \rangle$ and $\langle v_k \uparrow, v_0 \uparrow \rangle$. (2) $\langle v_j \uparrow, v_{j+1} \downarrow \rangle$ for $j = 1 \ldots k-1$ (if $k = 1$ then there are no such states). (3) In addition, for any v_j for $j = 1 \ldots k$ such that $v_j \in V_L$, add the states $\langle v_j \downarrow \rangle$ and $\langle v_j \uparrow \rangle$.
Step 2: Creating T: T is formed by including the following directed arcs: (1) Add an arc from $\langle a, b \rangle \in S$ to $\langle c, d \rangle \in S$ whenever $b = c$. (2) Add an arc from $\langle a, b \downarrow \rangle \in S$ to $\langle c \downarrow \rangle \in S$ whenever $b = c$. (3) Add an arc from $\langle a \uparrow \rangle \in S$ to $\langle b \uparrow, c \rangle \in S$ whenever $a = b$.

Figure 5-2: Algorithm for constructing a directed graph (S, T) from a set of hyperedges E.

Definition 1 A trigram path p *is* $p = \langle v_1, p_1, v_2, p_2, v_3 \rangle$ *where: a)* $v_1, v_2, v_3 \in V_L$*; b)* p_1 *is a path (sequence of states) between nodes* $\langle v_1 \uparrow \rangle$ *and* $\langle v_2 \downarrow \rangle$ *in the graph* (S, T)*; c)* p_2 *is a path between nodes* $\langle v_2 \uparrow \rangle$ *and* $\langle v_3 \downarrow \rangle$ *in the graph* (S, T)*. We define* $\mathcal{P}$ *to be the set of all trigram paths in* (S, T)*.*

The set $\mathcal{P}$ of trigram paths plays an analogous role to the set $\mathcal{B}$ of bigrams in our previous algorithm.

We use $v_1(p), p_1(p), v_2(p), p_2(p), v_3(p)$ to refer to the individual components of a path p. In addition, define S_N to be the set of states in S of the form $\langle a, b \rangle$ (as opposed to the form $\langle c \downarrow \rangle$ or $\langle c \uparrow \rangle$ where $c \in V_L$).

We now define a new index set, $\mathcal{I} = V \cup E \cup S_N \cup \mathcal{P}$, adding variables y_s for $s \in S_N$, and y_p for $p \in \mathcal{P}$. If we take $\mathcal{Y} \subset \{0, 1\}^{|\mathcal{I}|}$ to be the set of valid derivations, the optimization problem is to find $y^* = \arg\max_{y \in \mathcal{Y}} f(y)$, where $f(y) = \theta \cdot y$, that is,

$$f(y) = \sum_v \theta_v y_v + \sum_e \theta_e y_e + \sum_s \theta_s y_s + \sum_p \theta_p y_p$$

In particular, we might define $\theta_s = 0$ for all s, and $\theta_p = \log p(l(v_3(p)) | l(v_1(p)), l(v_2(p)))$ where $p(w_3 | w_1, w_2)$ is a trigram probability.

- **D0.** The y_v and y_e variables form a valid derivation in the original hypergraph.
- **D1.** For all $s \in S_N$, $y_s = \sum_{e:s \in S(e)} y_e$ (see figure 5-2 for the definition of $S(e)$).
- **D2.** For all $v \in V_L$, $y_v = \sum_{p:v_3(p)=v} y_p$
- **D3.** For all $v \in V_L$, $y_v = \sum_{p:v_2(p)=v} y_p$
- **D4.** For all $v \in V_L$, $y_v = \sum_{p:v_1(p)=v} y_p$
- **D5.** For all $s \in S_N$, $y_s = \sum_{p:s \in p_1(p)} y_p$
- **D6.** For all $s \in S_N$, $y_s = \sum_{p:s \in p_2(p)} y_p$
- Lagrangian with Lagrange multipliers for **D3**–**D6**:

$$\begin{aligned} L(y, \lambda, \gamma, u, v) = \theta \cdot y \\ + \sum_v \lambda_v \left(y_v - \sum_{p:v_2(p)=v} y_p \right) \\ + \sum_v \gamma_v \left(y_v - \sum_{p:v_1(p)=v} y_p \right) \\ + \sum_s u_s \left(y_s - \sum_{p:s \in p_1(p)} y_p \right) \\ + \sum_s v_s \left(y_s - \sum_{p:s \in p_2(p)} y_p \right). \end{aligned}$$

Figure 5-3: Constraints **D0**–**D6**, and the Lagrangian.

The set $\mathcal{P}$ is large (typically exponential in size): however, we will see that we do not need to represent the y_p variables explicitly. Instead we will be able to leverage the underlying structure of a path as a sequence of states.

The set of valid derivations is $\mathcal{Y} = \{y : y \text{ satisfies constraints } \mathbf{D0}\text{–}\mathbf{D6}\}$ where the constraints are shown in figure 5-3. **D1** simply states that $y_s = 1$ iff there is exactly one edge e in the derivation such that $s \in S(e)$. Constraints **D2**–**D4** enforce consistency between leaves in the trigram paths, and the y_v values. Constraints **D5** and **D6** enforce consistency between states seen in the paths, and the y_s values.

The Lagrangian relaxation algorithm is then derived in a similar way to before. Define

$$\mathcal{Y}' = \{y : y \text{ satisfies constraints } \mathbf{D0}\text{–}\mathbf{D2}\}$$

We have dropped the **D3**–**D6** constraints, but these will be introduced using Lagrange multipliers. The resulting Lagrangian is shown in figure 5-3, and can be written as $L(y, \lambda, \gamma, u, v) = \beta \cdot y$ where $\beta_v = \theta_v + \lambda_v + \gamma_v$, $\beta_s = \theta_s + u_s + v_s$, $\beta_p = \theta_p - \lambda(v_2(p)) - \gamma(v_1(p)) - \sum_{s \in p_1(p)} u(s) - \sum_{s \in p_2(p)} v(s)$.

Initialization: Set $\lambda^0 = 0, \gamma^0 = 0, u^0 = 0, v^0 = 0$
Algorithm: For $t = 1 \ldots T$:

- $y^t = \arg\max_{y \in \mathcal{Y}'} L(y, \lambda^{t-1}, \gamma^{t-1}, u^{t-1}, v^{t-1})$
- If y^t satisfies the constraints **D3**–**D6**, return y^t, else:
  - $\forall v \in V_L$, $\lambda_v^t = \lambda_v^{t-1} - \delta^t(y_v^t - \sum_{p:v_2(p)=v} y_p^t)$
  - $\forall v \in V_L$, $\gamma_v^t = \gamma_v^{t-1} - \delta^t(y_v^t - \sum_{p:v_1(p)=v} y_p^t)$
  - $\forall s \in S_N$, $u_s^t = u_s^{t-1} - \delta^t(y_s^t - \sum_{p:s \in p_1(p)} y_p^t)$
  - $\forall s \in S_N$, $v_s^t = v_s^{t-1} - \delta^t(y_s^t - \sum_{p:s \in p_2(p)} y_p^t)$

Figure 5-4: The full Lagrangian relaxation algortihm. $\delta^t > 0$ is the step size at iteration t.

The dual is $L(\lambda, \gamma, u, v) = \max_{y \in \mathcal{Y}'} L(y, \lambda, \gamma, u, v)$; figure 5-4 shows a subgradient method that minimizes this dual. The key step in the algorithm at each iteration is to compute $\arg\max_{y \in \mathcal{Y}'} L(y, \lambda, \gamma, u, v) = \arg\max_{y \in \mathcal{Y}'} \beta \cdot y$ where β is defined above. Again, our definition of $\mathcal{Y}'$ allows this maximization to be performed efficiently, as follows:

1. For each $v \in V_L$, define $\alpha_v^* = \arg\max_{p:v_3(p)=v} \beta(p)$, and $\alpha_v = \beta(\alpha_v^*)$. (i.e., for each v, compute the highest scoring trigram path ending in v.)

2. Find values for the y_v, y_e and y_s variables that form a valid derivation, and that maximize
   $f'(y) = \sum_v (\beta_v + \alpha_v) y_v + \sum_e \beta_e y_e + \sum_s \beta_s y_s$

3. Set $y_p = 1$ iff $y_{v_3(p)} = 1$ and $p = \alpha_{v_3(p)}^*$.

The first step involves finding the highest scoring incoming trigram path for each leaf v. This step can be performed efficiently using the Floyd-Warshall all-pairs shortest path algorithm [Floyd, 1962] over the graph (S, T); the details are given in the appendix. The second step involves simple dynamic programming over the hypergraph (V, E) (it is simple to integrate the β_s terms into this algorithm). In the third step, the path variables y_p are filled in.

5.4.3 Properties

We now describe some important properties of the algorithm:

Efficiency. The main steps of the algorithm are: 1) construction of the graph (S, T); 2) at each iteration, dynamic programming over the hypergraph (V, E); 3) at each iteration, all-pairs shortest path algorithms over the graph (S, T). Each of these steps is vastly more efficient than computing an exact intersection of the hypergraph with a language model.

Exact solutions. By usual guarantees for Lagrangian relaxation, if at any point the algorithm returns a solution y^t that satisfies constraints **D3–D6**, then y^t exactly solves the problem in Eq. 5.1.

Upper bounds. At each point in the algorithm, $L(\lambda^t, \gamma^t, u^t, v^t)$ is an upper bound on the score of the optimal primal solution, $f(y^*)$. Upper bounds can be useful in evaluating the quality of primal solutions from either our algorithm or other methods such as cube pruning.

Simplicity of implementation. Construction of the (S, T) graph is straightforward. The other steps—hypergraph dynamic programming, and all-pairs shortest path—are widely known algorithms that are simple to implement.

5.5 Tightening the Relaxation

The algorithm that we have described minimizes the dual function $L(\lambda, \gamma, u, v)$. By usual results for Lagrangian relaxation (e.g., see [Korte and Vygen, 2008]), L is the dual function for a particular LP relaxation arising from the definition of $\mathcal{Y}'$ and the additional constaints **D3–D6**. In some cases the LP relaxation has an integral solution, in which case the algorithm will return an optimal solution y^t.[7] In other cases, when the LP relaxation has a fractional solution, the subgradient algorithm will still converge to the minimum of L, but the primal solutions y^t will move between a number of solutions.

We now describe a method that incrementally adds hard constraints to the set $\mathcal{Y}'$, until the method returns an exact solution. For a given $y \in \mathcal{Y}'$, for any v with $y_v = 1$, we can recover the previous two leaves (the trigram ending in v) from either the path variables y_p, or the hypergraph variables y_e. Specifically, define $v_{-1}(v, y)$ to be the leaf preceding v in the trigram path p with $y_p = 1$ and $v_3(p) = v$, and $v_{-2}(v, y)$ to be the leaf two positions before v in the trigram path p with $y_p = 1$ and $v_3(p) = v$. Similarly, define $v'_{-1}(v, y)$ and $v'_{-2}(v, y)$ to be the preceding two leaves under the y_e variables. If the method has not

[7] Provided that the algorithm is run for enough iterations for convergence.

converged, these two trigram definitions may not be consistent. For a consistent solution, we require $v_{-1}(v, y) = v'_{-1}(v, y)$ and $v_{-2}(v, y) = v'_{-2}(v, y)$ for all v with $y_v = 1$. Unfortunately, explicitly enforcing all of these constraints would require exhaustive dynamic programming over the hypergraph using the [Bar-Hillel et al., 1964] method, something we wish to avoid.

Instead, we enforce a weaker set of constraints, which require far less computation. Assume some function $\pi : V_L \rightarrow \{1, 2, \ldots q\}$ that partitions the set of leaves into q different partitions. Then we will add the following constraints to $\mathcal{Y}'$:

$$\begin{aligned} \pi(v_{-1}(v, y)) &= \pi(v'_{-1}(v, y)) \\ \pi(v_{-2}(v, y)) &= \pi(v'_{-2}(v, y)) \end{aligned}$$

for all v such that $y_v = 1$. Finding $\arg\max_{y \in \mathcal{Y}'} \theta \cdot y$ under this new definition of $\mathcal{Y}'$ can be performed using the construction of [Bar-Hillel et al., 1964], with q different lexical items (for brevity we omit the details). This is efficient if q is small.[8]

The remaining question concerns how to choose a partition π that is effective in tightening the relaxation. To do this we implement the following steps: 1) run the subgradient algorithm until L is close to convergence; 2) then run the subgradient algorithm for m further iterations, keeping track of all pairs of leaf nodes that violate the constraints (i.e., pairs $a = v_{-1}(v, y)/b = v'_{-1}(v, y)$ or $a = v_{-2}(v, y)/b = v'_{-2}(v, y)$ such that $a \neq b$); 3) use a graph coloring algorithm to find a small partition that places all pairs $\langle a, b \rangle$ into separate partitions; 4) continue running Lagrangian relaxation, with the new constraints added. We expand π at each iteration to take into account new pairs $\langle a, b \rangle$ that violate the constraints.

In related work, Sontag et al. [Sontag et al., 2008] describe a method for inference in Markov random fields where additional constraints are chosen to tighten an underlying relaxation. Other relevant work in NLP includes [Tromble and Eisner, 2006, Riedel and Clarke, 2006]. Our use of partitions π is related to previous work on coarse-to-fine inference for machine translation [Petrov et al., 2008].

5.6 Experiments

We report experiments on translation from Chinese to English, using the tree-to-string model described in [Huang and Mi, 2010]. We use an identical model, and identical devel-

[8] In fact in our experiments we use the original hypergraph to compute admissible outside scores for an exact A* search algorithm for this problem. We have found the resulting search algorithm to be very efficient.

Time	%age (LR)	%age (DP)	%age (ILP)	%age (LP)
0.5s	37.5	10.2	8.8	21.0
1.0s	57.0	11.6	13.9	31.1
2.0s	72.2	15.1	21.1	45.9
4.0s	82.5	20.7	30.7	63.7
8.0s	88.9	25.2	41.8	78.3
16.0s	94.4	33.3	54.6	88.9
32.0s	97.8	42.8	68.5	95.2
Median time	0.79s	77.5s	12.1s	2.4s

Figure 5-5: Results showing percentage of examples that are decoded in less than t seconds, for $t = 0.5, 1.0, 2.0, \ldots, 32.0$. LR = Lagrangian relaxation; DP = exhaustive dynamic programming; ILP = integer linear programming; LP = linear programming (LP does not recover an exact solution). The (I)LP experiments were carried out using Gurobi, a high-performance commercial-grade solver.

opment and test data, to that used by Huang and Mi.[9] The translation model is trained on 1.5M sentence pairs of Chinese-English data; a trigram language model is used. The development data is the newswire portion of the 2006 NIST MT evaluation test set (616 sentences). The test set is the newswire portion of the 2008 NIST MT evaluation test set (691 sentences).

We ran the full algorithm with the tightening method described in section 5.5. We ran the method for a limit of 200 iterations, hence some examples may not terminate with an exact solution. Our method gives exact solutions on 598/616 development set sentences (97.1%), and 675/691 test set sentences (97.7%).

In cases where the method does not converge within 200 iterations, we can return the best primal solution y^t found by the algorithm during those iterations. We can also get an upper bound on the difference $f(y^*) - f(y^t)$ using $\min_t L(u_t)$ as an upper bound on $f(y^*)$. Of the examples that did not converge, the worst example had a bound that was 1.4% of $f(y^t)$ (more specifically, $f(y^t)$ was -24.74, and the upper bound on $f(y^*) - f(y^t)$ was 0.34).

Figure 5-5 gives information on decoding time for our method and two other exact decoding methods: integer linear programming (using constraints **D0**–**D6**), and exhaustive dynamic programming using the construction of [Bar-Hillel et al., 1964]. Our method is clearly the most efficient, and is comparable in speed to state-of-the-art decoding algorithms.

We also compare our method to cube pruning [Chiang, 2007, Huang and Chiang, 2007]. We reimplemented cube pruning in C++, to give a fair comparison to our method. Cube

[9]We thank Liang Huang and Haitao Mi for providing us with their model and data.

pruning has a parameter, b, dictating the maximum number of items stored at each chart entry. With $b = 50$, our decoder finds higher scoring solutions on 50.5% of all examples (349 examples), the cube-pruning method gets a strictly higher score on only 1 example (this was one of the examples that did not converge within 200 iterations). With $b = 500$, our decoder finds better solutions on 18.5% of the examples (128 cases), cube-pruning finds a better solution on 3 examples. The median decoding time for our method is 0.79 seconds; the median times for cube pruning with $b = 50$ and $b = 500$ are 0.06 and 1.2 seconds respectively.

Our results give a very good estimate of the percentage of search errors for cube pruning. A natural question is how large b must be before exact solutions are returned on almost all examples. Even at $b = 1000$, we find that our method gives a better solution on 95 test examples (13.7%).

Figure 5-5 also gives a speed comparison of our method to a linear programming (LP) solver that solves the LP relaxation defined by constraints **D0**–**D6**. We still see speed-ups, in spite of the fact that our method is solving a harder problem (it provides integral solutions). The Lagrangian relaxation method, when run without the tightening method of section 5.5, is solving a dual of the problem being solved by the LP solver. Hence we can measure how often the tightening procedure is absolutely necessary, by seeing how often the LP solver provides a fractional solution. We find that this is the case on 54.0% of the test examples: the tightening procedure is clearly important. Inspection of the tightening procedure shows that the number of partitions required (the parameter q) is generally quite small: 59% of examples that require tightening require $q \leq 6$; 97.2% require $q \leq 10$.

5.7 Conclusion

We have described a Lagrangian relaxation algorithm for exact decoding of syntactic translation models, and shown that it is significantly more efficient than other exact algorithms for decoding tree-to-string models. There are a number of possible ways to extend this work. Our experiments have focused on tree-to-string models, but the method should also apply to Hiero-style syntactic translation models [Chiang, 2007]. Additionally, our experiments used a trigram language model, however the constraints in figure 5-3 generalize to higher-order language models. Finally, our algorithm recovers the 1-best translation for a given input sentence; it should be possible to extend the method to find k-best solutions.

Chapter 6

Conclusion

6.1 Summary

In this thesis, we have presented Lagrangian relaxation algorithms for four different decoding challenges in natural language processing. These include:

- Combining two dynamic programming algorithms with shared structure over a single sentence.
- Optimizing a corpus-level objective with inter-sentence constraints that tie together sentence-level models.
- Finding the best parse in higher-order non-projective dependency parsing, an NP-hard optimization problem.
- Exactly decoding a synchronous grammar with intersected language model for syntactic machine translation.

For each algorithm, we described how to decompose the decoding problem into easier subproblems that we solved with simple combinatorial algorithms. For the first two problems, we showed how to use off-the-shelf solvers to decode the subproblems. By this method, we were able to find the exact solution to difficult model extensions by directly utilizing known algorithms. For the second two problems, we constructed novel subproblems, based on head automata and all-pairs shortest path, that exploited the structure of the problem and helped us find solutions efficiently.

The result of this work is a series of general algorithms for natural language tasks. These algorithms have several important practical and theoretical properties:

- The algorithms are simple. The first two algorithms use known solvers from problems in NLP. The second two utilize well-known combinatorial algorithms.
- The algorithms are efficient. We show that the Lagrangian relaxation algorithms are significantly faster than highly optimized exact methods such as dynamic programming and general-purpose linear programming solvers.
- The algorithms are often exact. For all tasks, we show that empirically, the relaxations often lead to an exact solution to the original problem. Furthermore, when the algorithm does not produce an exact solution, we can tighten the relaxation to encourage exactness.

These properties make Lagrangian relaxation a useful decoding method for many natural language tasks.

6.2 Future Work

There are several areas of future work to further extend the algorithms presented in this thesis. Within each chapter we explored future directions for specific NLP tasks. Here we highlight general directions in the application of Lagrangian relaxation.

One important question is how to use these techniques for variants of the decoding problems presented here. We focused on a particular decoding optimization, but there are other variants that are used in practice. For instance, n-best decoding is an important variant of the decoding that is widely used in NLP, e.g. for reranking and approximate minimum-bayes risk decoding. There has been work on n-best decoding for linear programming [Fromer and Globerson, 2009] that could potentially be incorporated into Lagrangian relaxation algorithms.

Another open implementation question is how to develop fast approximate relaxation algorithms for very difficult decoding problems. For certain problems, it may be difficult to find any subproblems that are fast to solve exactly. For these problems, we may want to use approximate methods at each iteration and minimize the dual using ϵ-subgradient descent. This technique is not guaranteed to find an exact solution, but can produce a bounded approximation.

Similarly, there are many problems where the LP relaxation includes very few optimal solutions. For these problems Lagrangian relaxation is unlikely to converge to an exact result. We employed a tightening method in Chapter 5 to help deal with this problem, but we may need to use a more extensive approach from the study of integer linear programming, such as branch-and-bound or branch-and-cut. Lagrangian relaxation can act as an efficient subroutine for these algorithms.

Finally, the work in this thesis is focused on the decoding problem during test time, but for structured prediction problems, inference also plays a crucial role in training. These algorithms could be used as is as a black-box inference algorithms during training. Unfortunately, it is unlikely that they would produce a large proportion of exact solutions during training. Martins et al. [2009c] experiment with training using fractional LP solutions with promising results. These fractional solution can be extracted directly from Lagrangian relaxation algorithms even if the algorithm does not converge.

Appendix A

Appendix

A.1 Fractional Solutions

In Chapter 2 we claimed that $\mathcal{Q}'$ formed an outer bound of the set $\mathcal{Q}$ We now give an example of a member (μ, ν) of the set $\mathcal{Q}' \backslash \text{conv}(\mathcal{Q})$, corresponding to a fractional extreme point of the polytope $\mathcal{Q}'$. Note that constructing such examples is non-trivial. We found this one by repeatedly solving the primal LP relaxation with random objectives. Recall that the constraints in $\mathcal{Q}'$ specify that $\mu \in \text{conv}(\mathcal{Y})$, $\nu \in \text{conv}(\mathcal{Z})$, and $\mu(i,t) = \nu(i,t)$ for all $(i,t) \in \mathcal{I}_{\text{uni}}$. Given that $\mu \in \text{conv}(\mathcal{Y})$, it must be a convex combination of 1 or more members of $\mathcal{Y}$; a similar property holds for ν. We define the example as follows. There are two possible parts of speech, A and B, and an additional non-terminal symbol X. The sentence is of length 3, $w_1\ w_2\ w_3$. Consider the case where ν is a convex combination of two tag sequences, each with weight 0.5, namely $w_1/A\ w_2/A\ w_3/A$ and $w_1/A\ w_2/B\ w_3/B$. Take μ to be a convex combination of two parses, with weight 0.5 each, namely: $(X(A\ w_1)(X(A\ w_2)(B\ w_3)))$ and $(X(A\ w_1)(X(B\ w_2)(A\ w_3)))$. It can be verified that we have $\mu(i,t) = \nu(i,t)$ for all (i,t)—the marginals for single tags for μ and ν agree—even though both μ and ν are fractional.

To demonstrate that this fractional solution is a vertex of the polytope, we now give parameter values that give this fractional solution as the arg max of the inference problem. For the tagging model, set $\theta(AA \rightarrow A, 3) = \theta(AB \rightarrow B, 3) = 0$, and all other parameters to be some negative value. For the parsing model, set $\theta(X \rightarrow A\ X, 1, 1, 3) = \theta(X \rightarrow A\ B, 2, 2, 3) = \theta(X \rightarrow B\ A, 2, 2, 3) = 0$, and all other rule parameters to be negative. Under these settings it can be verified that the fractional solution has value 0, while all integral solutions have a negative value, hence the arg max must be fractional.

A.2 Implementation Details

This appendix describes details of the algorithms in Chapters 2 and 4, specifically the choice of the step sizes α_k, and use of the $\gamma(i,j)$ parameters.

A.2.1 Choice of Step Sizes

For Chapter 2, we used the following step size in our experiments. First, we initialized α_0 to equal 0.5, a relatively large value. Then we defined $\alpha_k = \alpha_0 * 2^{-\eta_k}$, where η_k is the number of times that $L(u^{(k')}) > L(u^{(k'-1)})$ for $k' \leq k$. This learning rate drops at a rate of $1/2^t$, where t is the number of times that the dual increases from one iteration to the next. See [Koo et al., 2010] for a similar, but less aggressive step size used to solve a more heavily constrained task.

For Chapter 4, we used a different step size. First, define $\delta = f(z^{(1)}) - f(y^{(1)})$, where $(z^{(1)}, y^{(1)})$ is the output of the algorithm on the first iteration (note that we always have $\delta \geq 0$ since $f(z^{(1)}) = L(u^{(1)})$). Then define $\alpha_k = \delta/(1+\eta_k)$, where η_k is the number of times that $L(u^{(k')}) > L(u^{(k'-1)})$ for $k' \leq k$. Hence the learning rate drops at a rate of $1/(1+t)$, where t is the number of times that the dual increases from one iteration to the next.

A.2.2 Use of the $\gamma(i,j)$ Parameters

The parsing algorithms both consider a generalized problem that includes $\gamma(i,j)$ parameters. We now describe how these can be useful. Recall that the optimization problem is to solve $\arg\max_{z \in \mathcal{Z}, y \in \mathcal{Y}} f(z) + h(y)$, subject to a set of agreement constraints. In our models, $f(z)$ can be written as $f'(z) + \sum_{i,j} \alpha(i,j) z(i,j)$ where $f'(z)$ includes only terms depending on higher-order (non arc-factored features), and $\alpha(i,j)$ are weights that consider the dependency between i and j alone. For any value of $0 \leq \beta \leq 1$, the problem $\arg\max_{z \in \mathcal{Z}, y \in \mathcal{Y}} f_2(z) + h_2(y)$ is equivalent to the original problem, if $f_2(z) = f'(z) + (1-\beta) \sum_{i,j} \alpha(i,j) z(i,j)$ and $h_2(y) = \beta \sum_{i,j} \alpha(i,j) y(i,j)$. We have simply shifted the $\alpha(i,j)$ weights from one model to the other. While the optimization problem remains the same, the algorithms in Figure 4-1 and 4-2 will converge at different rates depending on the value for β. In our experiments we set $\beta = 0.001$, which puts almost all the weight in the head-automata models, but allows weights on spanning tree edges to break ties in MST inference in a sensible way. We suspect this is important in early iterations of the

algorithm, when many values for $u(i,j)$ or $v(i,j)$ will be zero, and where with $\beta = 0$ many spanning tree solutions $y^{(k)}$ would be essentially random, leading to very noisy updates to the $u(i,j)$ and $v(i,j)$ values. We have not tested other values for β.

A.3 Computing the Optimal Trigram Paths

The decoding algorithm in Chapter 5 assumes an efficient algorithm for computing the optimal trigram paths. We use this section to fill in the details of this algorithm.

For each $v \in V_L$, define $\alpha_v = \max_{p:v_3(p)=v} \beta(p)$, where $\beta(p) = h(v_1(p), v_2(p), v_3(p)) - \lambda_1(v_1(p)) - \lambda_2(v_2(p)) - \sum_{s \in p_1(p)} u(s) - \sum_{s \in p_2(p)} v(s)$. Here h is a function that computes language model scores, and the other terms involve Lagrange mulipliers. Our task is to compute α_v^* for all $v \in V_L$.

It is straightforward to show that the S, T graph is *acyclic*. This will allow us to apply shortest path algorithms to the graph, even though the weights $u(s)$ and $v(s)$ can be positive or negative.

For any pair $v_1, v_2 \in V_L$, define $\mathcal{P}(v_1, v_2)$ to be the set of paths between $\langle v_1 \uparrow \rangle$ and $\langle v_2 \downarrow \rangle$ in the graph S, T. Each path p gets a score $score_u(p) = -\sum_{s \in p} u(s)$. Next, define $p_u^*(v_1, v_2) = \arg\max_{p \in \mathcal{P}(v_1, v_2)} score_u(p)$, and $score_u^*(v_1, v_2) = score_u(p^*)$. We assume similar definitions for $p_v^*(v_1, v_2)$ and $score_v^*(v_1, v_2)$. The p_u^* and $score_u^*$ values can be calculated using an all-pairs shortest path algorithm, with weights $u(s)$ on nodes in the graph. Similarly, p_v^* and $score_v^*$ can be computed using all-pairs shortest path with weights $v(s)$ on the nodes.

Having calculated these values, define $\mathcal{T}(v)$ for any leaf v to be the set of trigrams (x, y, v) such that: 1) $x, y \in V_L$; 2) there is a path from $\langle x \uparrow \rangle$ to $\langle y \downarrow \rangle$ and from $\langle y \uparrow \rangle$ to $\langle v \downarrow \rangle$ in the graph S, T. Then we can calculate

$$\begin{aligned} \alpha_v = \max_{(x,y,v) \in \mathcal{T}(v)} & (h(x,y,v) - \lambda_1(x) - \lambda_2(y) \\ & + p_u^*(x,y) + p_v^*(y,v)) \end{aligned}$$

in $O(|\mathcal{T}(v)|)$ time, by brute force search through the set $\mathcal{T}(v)$.

Bibliography

H. Alshawi. Head Automata and Bilingual Tiling: Translation with Minimal Representations. In *Proc. ACL*, pages 167–176, 1996. doi: 10.3115/981863.981886. URL `http://www.aclweb.org/anthology/P96-1023`.

Y. Bar-Hillel, M. Perles, and E. Shamir. On formal properties of simple phrase structure grammars. In *Language and Information: Selected Essays on their Theory and Application*, pages 116–150, 1964.

S. Buchholz and E. Marsi. CoNLL-X Shared Task on Multilingual Dependency Parsing. In *Proc. CoNLL*, pages 149–164, 2006.

X. Carreras. Experiments with a Higher-Order Projective Dependency Parser. In *Proc. EMNLP-CoNLL*, pages 957–961, 2007a.

X. Carreras. Experiments with a higher-order projective dependency parser. In *Proc. CoNLL*, pages 957–961, 2007b.

X. Carreras, M. Collins, and T. Koo. TAG, dynamic programming, and the perceptron for efficient, feature-rich parsing. In *Proc CONLL*, pages 9–16, 2008.

E. Charniak and M. Johnson. Coarse-to-fine n-best parsing and MaxEnt discriminative reranking. In *Proc. ACL*, page 180, 2005.

D. Chiang. A hierarchical phrase-based model for statistical machine translation. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, pages 263–270. Association for Computational Linguistics, 2005.

D. Chiang. Hierarchical phrase-based translation. *computational linguistics*, 33(2):201–228, 2007. ISSN 0891-2017.

M. Collins. Discriminative Training Methods for Hidden Markov Models: Theory and Experiments with Perceptron Algorithms. In *Proc. EMNLP*, pages 1–8, 2002a.

M. Collins. Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In *Proc. EMNLP*, page 8, 2002b.

M. Collins. Head-driven statistical models for natural language parsing. In *Computational linguistics*, volume 29, pages 589–637, 2003.

J. Duchi, D. Tarlow, G. Elidan, and D. Koller. Using Combinatorial Optimization within Max-Product Belief Propagation. In *NIPS*, pages 369–376, 2007.

J. Eisner. Bilexical grammars and their cubic-time parsing algorithms. *Advances in Probabilistic and Other Parsing Technologies*, pages 29–62, 2000a.

J. Eisner. Bilexical grammars and their cubic-time parsing algorithms. In *Advances in Probabilistic and Other Parsing Technologies*, pages 29–62, 2000b.

P. Felzenszwalb and D. McAllester. The generalized A* architecture. 29(153-190):2Update, 2007.

J.R. Finkel, T. Grenager, and C. Manning. Incorporating non-local information into information extraction systems by gibbs sampling. In *ANNUAL MEETING-ASSOCIATION FOR COMPUTATIONAL LINGUISTICS*, volume 43, page 363, 2005.

T. Finley and T. Joachims. Training structural svms when exact inference is intractable. In *ICML*, pages 304–311, 2008. ISBN 978-1-60558-205-4. doi: http://doi.acm.org/10.1145/1390156.1390195.

Robert W. Floyd. Algorithm 97: Shortest path. *Commun. ACM*, 5:345, 1962. ISSN 0001-0782.

M. Fromer and A. Globerson. An lp view of the m-best map problem. *Advances in Neural Information Processing Systems*, 22:567–575, 2009.

A. Globerson and T. Jaakkola. Fixing max-product: Convergent message passing algorithms for MAP LP-relaxations. In *NIPS*, volume 21, 2007.

David Graff. North american news text corpus. *Linguistic Data Consortium*, LDC95T21, 1995.

Michael Held and Richard M. Karp. The traveling-salesman problem and minimum spanning trees: Part ii. *Mathematical Programming*, 1:6–25, 1971. ISSN 0025-5610. URL `http://dx.doi.org/10.1007/BF01584070`. 10.1007/BF01584070.

Liang Huang and David Chiang. Forest rescoring: Faster decoding with integrated language models. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 144–151, Prague, Czech Republic, June 2007. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/P07-1019`.

Liang Huang and Haitao Mi. Efficient incremental decoding for tree-to-string translation. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 273–283, Cambridge, MA, October 2010. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/D10-1027`.

A.K. Joshi and Y. Schabes. Tree-Adjoining Grammars. *Handbook of Formal Languages: Beyond Words*, 3:69–123, 1997.

D. Klein and C.D. Manning. A* parsing: Fast exact Viterbi parse selection. In *Proc. of HLT-NAACL*, volume 3, pages 119–126, 2003.

K. Knight. Decoding complexity in word-replacement translation models. *Computational Linguistics*, 25(4):607–615, 1999.

N. Komodakis, N. Paragios, and G. Tziritas. MRF Optimization via Dual Decomposition: Message-Passing Revisited. In *Proc. ICCV*, 2007.

T. Koo, A. Globerson, X. Carreras, and M. Collins. Structured Prediction Models via the Matrix-Tree Theorem. In *Proc. EMNLP-CoNLL*, pages 141–150, 2007.

T. Koo, X. Carreras, and M. Collins. Simple semi-supervised dependency parsing. In *Proc. ACL/HLT*, 2008.

Terry Koo, Alexander M. Rush, Michael Collins, Tommi Jaakkola, and David Sontag. Dual decomposition for parsing with non-projective head automata. In *EMNLP*, 2010. URL `http://www.aclweb.org/anthology/D10-1125`.

B.H. Korte and J. Vygen. *Combinatorial Optimization: Theory and Algorithms*. Springer Verlag, 2008.

A. Kulesza and F. Pereira. Structured learning with approximate inference. In *NIPS*. 2008.

I. Langkilde. Forest-based statistical sentence generation. In *Proceedings of the 1st North American chapter of the Association for Computational Linguistics conference*, pages 170–177. Morgan Kaufmann Publishers Inc., 2000.

Daniel Marcu, Wei Wang, Abdessamad Echihabi, and Kevin Knight. Spmt: Statistical machine translation with syntactified target language phrases. In *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing*, pages 44–52, Sydney, Australia, July 2006. Association for Computational Linguistics.

M.P. Marcus, B. Santorini, and M.A. Marcinkiewicz. Building a large annotated corpus of English: The Penn Treebank. In *Computational linguistics*, volume 19, pages 313–330, 1994.

R.K. Martin, R.L. Rardin, and B.A. Campbell. Polyhedral characterization of discrete dynamic programming. *Operations research*, 38(1):127–138, 1990.

A.F.T. Martins, D. Das, N.A. Smith, and E.P. Xing. Stacking Dependency Parsers. In *Proc. EMNLP*, pages 157–166, 2008.

A.F.T. Martins, N.A. Smith., and E.P. Xing. Concise Integer Linear Programming Formulations for Dependency Parsing. In *Proc. ACL*, pages 342–350, 2009a. URL `http://www.aclweb.org/anthology/P/P09/P09-1039`.

A.F.T. Martins, N.A. Smith, and E.P. Xing. Concise integer linear programming formulations for dependency parsing. In *Proc. ACL*, 2009b.

A.F.T. Martins, N.A. Smith, and E.P. Xing. Polyhedral outer approximations with application to natural language parsing. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 713–720. ACM, 2009c.

R. McDonald and F. Pereira. Online Learning of Approximate Dependency Parsing Algorithms. In *Proc. EACL*, pages 81–88, 2006.

R. McDonald and G. Satta. On the Complexity of Non-Projective Data-Driven Dependency Parsing. In *Proc. IWPT*, 2007.

R. McDonald, F. Pereira, K. Ribarov, and J. Hajič. Non-Projective Dependency Parsing using Spanning Tree Algorithms. In *Proc. HLT-EMNLP*, pages 523–530, 2005a.

Ryan T. McDonald, Fernando Pereira, Kiril Ribarov, and Jan Hajic. Non-projective dependency parsing using spanning tree algorithms. In *HLT/EMNLP*, 2005b.

O. Meshi, D. Sontag, T. Jaakkola, and A. Globerson. Learning Efficiently with Approximate Inference via Dual Losses. In *Proc. ICML*, 2010.

I. Meza-Ruiz and S. Riedel. Jointly identifying predicates, arguments and senses using markov logic. In *Proc. NAACL*, pages 155–163. Association for Computational Linguistics, 2009.

Angelia Nedić and Asuman Ozdaglar. Approximate primal solutions and rate analysis for dual subgradient methods. *SIAM Journal on Optimization*, 19(4):1757–1780, 2009.

J. Nivre and R. McDonald. Integrating Graph-Based and Transition-Based Dependency Parsers. In *Proc. ACL*, pages 950–958, 2008.

Slav Petrov, Aria Haghighi, and Dan Klein. Coarse-to-fine syntactic machine translation using language projections. In *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing*, pages 108–116, Honolulu, Hawaii, October 2008. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/D08-1012`.

V. Punyakanok, D. Roth, W. Yih, and D. Zimak. Learning and Inference over Constrained Output. In *Proc. IJCAI*, pages 1124–1129. 2005.

Sebastian Riedel and James Clarke. Incremental integer linear programming for non-projective dependency parsing. In *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing*, EMNLP '06, pages 129–137, Stroudsburg, PA, USA, 2006. Association for Computational Linguistics.

A.M. Rush, D. Sontag, M. Collins, and T. Jaakkola. On Dual Decomposition and Linear Programming Relaxations for Natural Language Processing. In *Proc. EMNLP*, 2010.

Libin Shen, Jinxi Xu, and Ralph Weischedel. A new string-to-dependency machine translation algorithm with a target dependency language model. In *Proceedings of ACL-08: HLT*, pages 577–585, Columbus, Ohio, June 2008. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/P/P08/P08-1066`.

D.A. Smith and J. Eisner. Dependency Parsing by Belief Propagation. In *Proc. EMNLP*, pages 145–156, 2008a. URL `http://www.aclweb.org/anthology/D08-1016`.

D.A. Smith and J. Eisner. Dependency parsing by belief propagation. In *Proc. EMNLP*, pages 145–156, 2008b.

D.A. Smith and N.A. Smith. Probabilistic Models of Nonprojective Dependency Trees. In *Proc. EMNLP-CoNLL*, pages 132–140, 2007.

D. Sontag, T. Meltzer, A. Globerson, T. Jaakkola, and Y. Weiss. Tightening LP relaxations for MAP using message passing. In *Proc. UAI*, 2008.

M. Steedman. *The Syntactic Process.* MIT Press, 2000.

M. Surdeanu, R. Johansson, A. Meyers, L. Màrquez, and J. Nivre. The CoNLL-2008 Shared Task on Joint Parsing of Syntactic and Semantic Dependencies. In *Proc. CoNLL*, 2008.

B. Taskar, C. Guestrin, and D. Koller. Max-margin Markov networks. In *NIPS*, 2003.

B. Taskar, D. Klein, M. Collins, D. Koller, and C. Manning. Max-margin parsing. In *Proc. EMNLP*, pages 1–8, 2004.

K. Toutanova and C.D. Manning. Enriching the knowledge sources used in a maximum entropy part-of-speech tagger. In *Proc. EMNLP*, pages 63–70, 2000.

Kristina Toutanova, Dan Klein, Christopher D. Manning, and Yoram Singer. Feature-rich part-of-speech tagging with a cyclic dependency network. In *HLT-NAACL*, 2003.

Roy W. Tromble and Jason Eisner. A fast finite-state relaxation method for enforcing global constraints on sequence decoding. In *Proceedings of the main conference on Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics*, HLT-NAACL '06, pages 423–430, Stroudsburg, PA, USA, 2006. Association for Computational Linguistics.

M. Wainwright and M. I. Jordan. *Graphical Models, Exponential Families, and Variational Inference.* Now Publishers Inc., Hanover, MA, USA, 2008. ISBN 1601981848, 9781601981844.

M. Wainwright, T. Jaakkola, and A. Willsky. MAP estimation via agreement on trees: message-passing and linear programming. In *IEEE Transactions on Information Theory*, volume 51, pages 3697–3717, 2005a.

M. Wainwright, T. Jaakkola, and A. Willsky. A new class of upper bounds on the log partition function. In *IEEE Transactions on Information Theory*, volume 51, pages 2313–2335, 2005b.